Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning

Matthew Jagielski*, Alina Oprea*, Battista Biggio [†], Chang Liu[‡], Cristina Nita-Rotaru*, and Bo Li[‡]

*Northeastern University, Boston, MA   [†]Univ. Cagliary, Cagliary, Italy   [‡]UC Berkeley, Berkeley, CA

Abstract—As machine learning becomes widely used for auto-mated decisions, attackers have strong incentives to manipulate the results and models generated by machine learning algorithms. In this paper, we perform the first systematic study of poisoning attacks and their countermeasures for linear regression models. In poisoning attacks, attackers deliberately influence the training data to manipulate the results of a predictive model. We propose a theoretically-grounded optimization framework specifically designed for linear regression and demonstrate its effectiveness on a range of datasets and models. We also introduce a fast statistical attack that requires limited knowledge of the training process. Finally, we design a new principled defense method that is highly resilient against all poisoning attacks. We provide formal guarantees about its convergence and an upper bound on the effect of poisoning attacks when the defense is deployed. We evaluate extensively our attacks and defenses on three realistic datasets from health care, loan assessment, and real estate domains.

I. INTRODUCTION

As more applications with large societal impact rely on machine learning for automated decisions, several concerns have emerged about potential vulnerabilities introduced by machine learning algorithms. Sophisticated attackers have strong incentives to manipulate the results and models generated by machine learning algorithms to achieve their objectives. For instance, attackers can deliberately influence the training dataset to manipulate the results of a predictive model (in *poisoning* attacks [5], [39]–[41], [44], [46], [54]), cause mis-classification of new data in the testing phase (in *evasion attacks* [3], [8], [20], [42], [43], [49], [50]) or infer private information on training data (in *privacy attacks* [18], [19], [47]). Several experts from academia and industry highlighted the importance of considering these vulnerabilities in design-ing machine learning systems in a recent hearing held by the Senate Subcommittee on Space, Science, and Competitiveness entitled "The Dawn of AI" [23]. The field of *adversarial machine learning* studies the effect of such attacks against machine learning models and aims to design robust defense algorithms [25]. A comprehensive survey can be found in [6].

We consider the setting of poisoning attacks here, in which attackers inject a small number of corrupted points in the training process. Such poisoning attacks have been practically demonstrated in worm signature generation [41], [44], spam filters [39], DoS attack detection [46], PDF malware classifi-cation [54], handwritten digit recognition [5], and sentiment analysis [40]. We argue that these attacks become easier to mount today as many machine learning models need to be updated regularly to account for continuously-generated data. Such scenarios require *online training*, in which machine learning models are updated based on new incoming training data. For instance, in cyber-security analytics, new Indicators of Compromise (IoC) rise due to the natural evolution of malicious threats, resulting in updates to machine learning models for threat detection [22]. These IoCs are collected from online platforms like VirusTotal, in which attackers can also submit IoCs of their choice. In personalized medicine, it is envisioned that patient treatment is adjusted in real-time by analyzing information crowdsourced from multiple participants [15]. By controlling a few devices, attackers can submit fake information (e.g., sensor measurements), which is then used for training models applied to a large set of pa-tients. Defending against such poisoning attacks is challenging with current techniques. Methods from robust statistics (e.g, [17], [26]) are resilient against noise but perform poorly on adversarially-poisoned data, and methods for sanitization of training data operate under restrictive adversarial models [12].

One fundamental class of supervised learning is linear regression. Regression is widely used for prediction in many settings (e.g., insurance or loan risk estimation, personalized medicine, market analysis). In a regression task a numerical *response variable* is predicted using a number of *predictor variables*, by learning a model that minimizes a *loss function*. Regression is powerful as it can also be used for classification tasks by mapping numerical predicted values into class labels. Assessing the real impact of adversarial manipulation of training data in linear regression, as well as determining how to design learning algorithms resilient under strong adversarial models is not yet well understood.

In this paper, we conduct the first systematic study of poisoning attacks and their countermeasures for linear re-gression models. We make the following contributions: (1) we are the first to consider the problem of poisoning linear regression under different adversarial models; (2) starting from an existing baseline poisoning attack for classification, we propose a theoretically-grounded optimization framework specifically tuned for regression models; (3) we design a fast statistical attack that requires minimal knowledge on the learning process; (4) we propose a principled defense algorithm with significantly increased robustness than known methods against a large class of attacks; (5) we extensively

evaluate our attacks and defenses on four regression models (OLS, LASSO, ridge, and elastic net), and on several datasets from different domains, including health care, loan assessment, and real estate. We elaborate our contributions below.

• On the attack dimension, we are the first to consider the problem of poisoning attacks against linear regression models. Compared to classification poisoning, in linear regression the response variables are continuous and their values also can be selected by the attacker. First, we adapt an existing poisoning attack for classification [54] into a baseline regression attack. Second, we design an optimization framework for regression poisoning in which the initialization strategy, the objective function, and the optimization variables can be selected to maximize the attack's impact on a particular model and dataset. Third, we introduce a fast statistical attack that is motivated by our theoretical analysis and insights. We find that optimization-based attacks are in general more effective than statistical-based techniques, at the expense of higher computational overhead and more information required by the adversary on the training process.

• On the defense axis, we propose a principled approach to constructing a defense algorithm called TRIM, which provides high robustness and resilience against a large class of poisoning attacks. The TRIM method estimates the regression parameters iteratively, while using a trimmed loss function to remove points with large residuals. After few iterations, TRIM is able to isolate most of the poisoning points and learn a robust regression model. TRIM performs significantly better and is much more effective in providing robustness compared to known methods from robust statistics (Huber [26] and RANSAC [17]), typically designed to provide resilience against noise and outliers. In contrast to these methods, TRIM is resilient to poisoned points with similar distribution as the training set. TRIM also outperforms other robust regression algorithms designed for adversarial settings (e.g., Chen et al. [11] and RONI [39]). We provide theoretical guarantees on the convergence of the algorithm and an upper bound on the model Mean Squared Error (MSE) generated when a fixed percentage of poisoned data is included in the training set.

• We evaluate our novel attacks and defenses extensively on four linear regression models and three datasets from health care, loan assessment, and real estate domains. First, we demonstrate the significant improvement of our attacks over the baseline attack of Xiao et al. in poisoning all models and datasets. For instance, the MSEs of our attacks are increased **by a factor of 6.83** compared to the Xiao et al. attack, and **a factor of 155.7** compared to unpoisoned regression models. In a case study health application, we find that our attacks can cause devastating consequences. The optimization attack causes 75% of patients' Warfarin medicine dosages to change by an average of 93.49%, while one tenth of these patients have their dosages changed by 358.89%. Second, we show that our defense TRIM is also significantly more robust than existing methods against all the attacks we developed. TRIM achieves MSEs **within 1%** of the unpoisoned model MSEs.

TRIM achieves MSEs much lower than existing methods, improving Huber by a factor of 1295.45, RANSAC by a factor of 75, and RONI by a factor of 71.13.

Outline. We start by providing background on regression learning, as well as introducing our system and adversarial model in Section II. We describe the baseline attack adapted from Xiao et al. [54], and our new poisoning attacks in Section III. Subsequently, we introduce our novel defense algorithm TRIM in Section IV. Section V includes a detailed experimental analysis of our attacks and defenses, as well as comparison with previous methods. Finally, we present related work in Section VI and conclude in Section VII.

II. System and adversarial model

Linear regression is at the basis of machine learning [24]. It is widely studied and applied in many applications due to its efficiency, simplicity of use, and effectiveness. Other more advanced learning methods (e.g., logistic regression, SVM, neural networks) can be seen as generalizations or extensions of linear regression. We systematically study the effect of poisoning attacks and their defenses for linear regression. We believe that our understanding of the resilience of this fundamental class of learning model to adversaries will enable future research on other classes of supervised learning methods.

Problem definition. Our system model is a supervised setting consisting of a *training phase* and a *testing phase* as shown in Figure 1 on the left ("Ideal world"). The learning process includes a data pre-processing stage that performs data cleaning and normalization, after which the training data can be represented, without loss of generality, as $\mathcal{D}_{\mathrm{tr}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$, where $\boldsymbol{x}_i \in [0,1]^d$ are d-dimensional numerical *predictor variables* (or *feature vectors*) and $y_i \in [0,1]$ are numerical *response variables*, for $i \in \{1, \ldots, n\}$. After that, the learning algorithm is applied to generate the *regression model* at the end of the training phase. In the testing phase, the model is applied to new data after pre-processing, and a numerical predicted value is generated using the regression model learned in training. Our model thus captures a standard multi-dimensional regression setting applicable to different prediction tasks.

In linear regression, the model output at the end of the training stage is a linear function $f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{w}^{\top} \boldsymbol{x} + b$, which predicts the value of y at $\boldsymbol{x}$. This function is parametrized by a vector $\boldsymbol{\theta} = (\boldsymbol{w}, b) \in \mathbb{R}^{d+1}$ consisting of the feature weights $\boldsymbol{w} \in \mathbb{R}^d$ and the bias $b \in \mathbb{R}$. Note that regression is substantially different from classification, as the y values are numerical, rather than being a set of indices (each denoting a different class from a predetermined set). The parameters of f are chosen to minimize a quadratic loss function:

$$\mathcal{L}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta}) = \underbrace{\frac{1}{n} \sum_{i=1}^{n} \left(f(\boldsymbol{x}_i, \boldsymbol{\theta}) - y_i \right)^2}_{\mathrm{MSE}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta})} + \lambda \Omega(\boldsymbol{w}), \quad (1)$$

where the Mean Squared Error $\mathrm{MSE}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta})$ measures the error in the predicted values assigned by $f(\cdot, \boldsymbol{\theta})$ to the training samples in $\mathcal{D}_{\mathrm{tr}}$ as the sum of squared residuals, $\Omega(\boldsymbol{w})$ is a regularization term penalizing large weight values, and λ
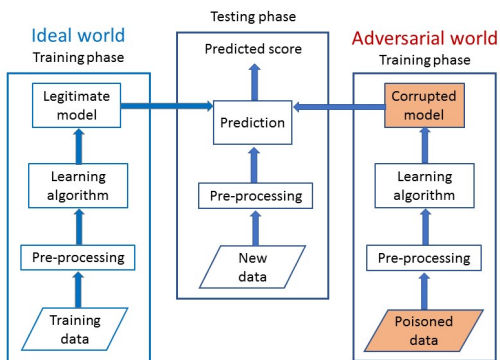
Fig. 1: System architecture.

is the so-called regularization parameter. Regularization is used to prevent *overfitting*, i.e., to preserve the ability of the learning algorithm to *generalize* well on unseen (testing) data. For regression problems, this capability, i.e., the expected performance of the trained function f on unseen data, is typically assessed by measuring the MSE on a separate test set. Popular linear regression methods differ mainly in the choice of the regularization term. In particular, we consider four models in this paper:

1) **Ordinary Least Squares (OLS)**, for which $\Omega(\boldsymbol{w}) = 0$ (i.e., no regularization is used);
2) **Ridge regression**, which uses ℓ_2-norm regularization $\Omega(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$;
3) **LASSO**, which uses ℓ_1-norm regularization $\Omega(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$;
4) **Elastic-net regression**, which uses a combination of ℓ_1-norm and ℓ_2-norm regularization $\Omega(\boldsymbol{w}) = \rho\|\boldsymbol{w}\|_1 + (1-\rho)\frac{1}{2}\|\boldsymbol{w}\|_2^2$, where $\rho \in (0,1)$ is a configurable parameter, commonly set to 0.5 (as we do in this work).

When designing a poisoning attack, we consider two metrics for quantifying the effectiveness of the attack. First, we measure the success rate of the poisoning attack by the difference in testing set MSE of the corrupted model compared to the legitimate model (trained without poisoning). Second, we consider the running time of the attack.

A. Adversarial model

We provide here a detailed adversarial model for poisoning attacks against regression algorithms, inspired from previous work in [4], [25], [38], [54]. The model consists of defining the adversary's goal, knowledge of the attacked system, and capability of manipulating the training data, to eventually define an optimal poisoning attack strategy.

Adversary's Goal. The goal of the attacker is to corrupt the learning model generated in the training phase, so that predictions on new data will be modified in the testing phase. The attack is considered a *poisoning availability attack*, if its goal is to affect prediction results indiscriminately, i.e., to cause a denial of service. It is instead referred to as a *poisoning integrity attack*, if the goal is to cause specific mis-predictions

at test time, while preserving the predictions on the other test samples. This is a similar setting to that of backdoor poisoning attacks recently reported in classification settings [9], [21].

Adversary's Knowledge. We assume here two distinct attack scenarios, referred to as *white-box* and *black-box* attacks in the following. In *white-box attacks*, the attacker is assumed to know the training data $\mathcal{D}_{\text{tr}}$, the feature values $\boldsymbol{x}$, the learning algorithm $\mathcal{L}$, and even the trained parameters $\boldsymbol{\theta}$. These attacks have been widely considered in previous work, although mainly against classification algorithms [5], [36], [54]. In *black-box attacks*, the attacker has no knowledge of the training set $\mathcal{D}_{\text{tr}}$ but can collect a substitute data set $\mathcal{D}'_{\text{tr}}$. The feature set and learning algorithm are known, while the trained parameters are not. However, the latter can be estimated by optimizing $\mathcal{L}$ on the substitute data set $\mathcal{D}'_{\text{tr}}$. This setting is useful to evaluate the *transferability* of poisoning attacks across different *training sets*, as discussed in [38], [54].

Adversary's Capability. In poisoning attacks, the attacker injects poisoning points into the training set before the regression model is trained (see the right side of Figure 1 labeled "Adversarial world"). The attacker's capability is normally limited by upper bounding the number p of *poisoning points* that can be injected into the training data, whose feature values and response variables are arbitrarily set by the attacker within a specified range (typically, the range covered by the training data, i.e., $[0,1]$ in our case) [38], [54]. The total number of points in the poisoned training set is thus $N = n + p$, with n being the number of pristine training samples. We then define the ratio $\alpha = p/n$, and the *poisoning rate* as the actual fraction of the training set controlled by the attacker, i.e., $n/N = \alpha/(1+\alpha)$. In previous work, poisoning rates higher than 20% have been only rarely considered, as the attacker is typically assumed to be able to control only a *small* fraction of the training data. This is motivated by application scenarios such as crowdsourcing and network traffic analysis, in which attackers can only reasonably control a *small* fraction of participants and network packets, respectively. Moreover, learning a sufficiently-accurate regression function in the presence of higher poisoning rates would be an ill-posed task, if not infeasible at all [5], [25], [36], [38], [53], [54].

Poisoning Attack Strategy. All the aforementioned poisoning attack scenarios, encompassing availability and integrity violations under white-box or black-box knowledge assumptions, can be formalized as a *bilevel optimization* problem [36], [38]. For white-box attacks, this can be written as:

$$\arg\max_{\mathcal{D}_p} \quad \mathcal{W}(\mathcal{D}', \boldsymbol{\theta}_p^\star), \qquad (2)$$
$$\text{s.t.} \quad \boldsymbol{\theta}_p^\star \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}_{\text{tr}} \cup \mathcal{D}_p, \boldsymbol{\theta}). \qquad (3)$$

The outer optimization amounts to selecting the poisoning points $\mathcal{D}_p$ to maximize a loss function $\mathcal{W}$ on an untainted data set $\mathcal{D}'$ (e.g., a validation set which does not contain any poisoning points), while the inner optimization corresponds to retraining the regression algorithm on a *poisoned* training set including $\mathcal{D}_p$. It should be clear that $\boldsymbol{\theta}_p^\star$ depends *implicitly* on the set $\mathcal{D}_p$ of poisoning attack samples through the solution of

the inner optimization problem. In poisoning integrity attacks, the attacker's loss $\mathcal{W}$ can be evaluated only on the points of interest (for which the attacker aims to cause mis-predictions at test time), while in poisoning availability attacks it is computed on an untainted set of data points, indiscriminately. In the black-box setting, the poisoned regression parameters $\boldsymbol{\theta}_p^\star$ are estimated using the substitute training data $\mathcal{D}'_{\mathrm{tr}}$ instead of $\mathcal{D}_{\mathrm{tr}}$.

In the remainder of this work, we only focus on poisoning availability attacks against regression learning, and on defending against them, as those have been mainly investigated in the literature of poisoning attacks. We highlight anyway again that poisoning integrity attacks can be implemented using the same technical derivation presented in this work, and leave a more detailed investigation of their effectiveness to future work.

III. ATTACK METHODOLOGY

In this section, we first discuss previously-proposed gradient-based optimization approaches to solving Problem (2)-(3) in classification settings. In Sect. III-A, we discuss how to adapt them to the case of regression learning, and propose novel strategies to further improve their effectiveness. Notably, since these attacks have been originally proposed in the context of classification problems, the class label of the attack sample is arbitrarily initialized and then kept fixed during the optimization procedure (recall that y is a categorical variable in classification). As we will demonstrate in the remainder of this work, a significant improvement we propose here to the current attack derivation is to *simultaneously* optimize the response variable of each poisoning point along with its feature values. We subsequently highlight some theoretical insights on how each poisoning sample is updated during the gradient-based optimization process. This will lead us to develop a much faster attack, presented in Sect. III-B, which only leverages some statistical properties of the data and requires minimal black-box access to the targeted model.

A. Optimization-based Poisoning Attacks

Previous work has considered solving Problem (2)-(3) by iteratively optimizing one poisoning sample at a time through gradient ascent [5], [36], [38], [54]. An exemplary algorithm is given as Algorithm 1. We denote with $\boldsymbol{x}_c$ the feature vector of the attack point being optimized, and with y_c its response variable (categorical for classification problems). In particular, in each iteration, the algorithm optimizes all points in $\mathcal{D}_p$, by updating their feature vectors one at a time. As reported in [54], the vector $\boldsymbol{x}_c$ can be updated through a line search along the direction of the gradient $\nabla_{\boldsymbol{x}_c}\mathcal{W}$ of the outer objective $\mathcal{W}$ (evaluated at the current poisoned solution) with respect to the poisoning point $\boldsymbol{x}_c$ (cf. line 7 in Algorithm 1). Note that this update step should also enforce $\boldsymbol{x}_c$ to lie within the feasible domain (e.g., $\boldsymbol{x}_c \in [0, 1]^d$), which can be typically achieved through simple projection operators [5], [38], [54]. The algorithm terminates when no sensible change in the outer objective $\mathcal{W}$ is observed.

Gradient Computation. The aforementioned algorithm is essentially a standard gradient-ascent algorithm with line search.

Algorithm 1 Poisoning Attack Algorithm

Input: $\mathcal{D} = \mathcal{D}_{\mathrm{tr}}$ (white-box) or $\mathcal{D}'_{\mathrm{tr}}$ (black-box), $\mathcal{D}'$, $\mathcal{L}$, $\mathcal{W}$, the initial poisoning attack samples $\mathcal{D}_p^{(0)} = (\boldsymbol{x}_c, y_c)_{c=1}^p$, a small positive constant ε.

1: $i \leftarrow 0$ (iteration counter)
2: $\boldsymbol{\theta}^{(i)} \leftarrow \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D} \cup \mathcal{D}_p^{(i)}, \boldsymbol{\theta})$
3: **repeat**
4: $\quad w^{(i)} \leftarrow \mathcal{W}(\mathcal{D}', \boldsymbol{\theta}^{(i)})$
5: $\quad \boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)}$
6: $\quad$ **for** c $= 1, \ldots, p$ **do**
7: $\quad\quad \boldsymbol{x}_c^{(i+1)} \leftarrow$ line_search $\left(\boldsymbol{x}_c^{(i)}, \nabla_{\boldsymbol{x}_c}\mathcal{W}(\mathcal{D}', \boldsymbol{\theta}^{(i+1)}) \right)$
8: $\quad\quad \boldsymbol{\theta}^{(i+1)} \leftarrow \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D} \cup \mathcal{D}_p^{(i+1)}, \boldsymbol{\theta})$
9: $\quad\quad w^{(i+1)} \leftarrow \mathcal{W}(\mathcal{D}', \boldsymbol{\theta}^{(i+1)})$
10: $\quad i \leftarrow i + 1$
11: **until** $|w^{(i)} - w^{(i-1)}| < \varepsilon$

Output: the final poisoning attack samples $\mathcal{D}_p \leftarrow \mathcal{D}_p^{(i)}$

The challenging part is understanding how to compute the required gradient $\nabla_{\boldsymbol{x}_c}\mathcal{W}(\mathcal{D}', \boldsymbol{\theta})$, as this has to capture the implicit dependency of the parameters $\boldsymbol{\theta}$ of the inner problem on the poisoning point $\boldsymbol{x}_c$. Indeed, assuming that $\mathcal{W}$ does not depend directly on $\boldsymbol{x}_c$, but only through $\boldsymbol{\theta}$, we can compute $\nabla_{\boldsymbol{x}_c}\mathcal{W}(\mathcal{D}', \boldsymbol{\theta})$ using the chain rule as:

$$\nabla_{\boldsymbol{x}_c}\mathcal{W} = \nabla_{\boldsymbol{x}_c}\boldsymbol{\theta}(\boldsymbol{x}_c)^\top \cdot \nabla_{\boldsymbol{\theta}}\mathcal{W}, \qquad (4)$$

where we have made explicit that $\boldsymbol{\theta}$ *depends* on $\boldsymbol{x}_c$. While the second term is simply the derivative of the outer objective with respect to the regression parameters, the first one captures the dependency of the solution $\boldsymbol{\theta}$ of the learning problem on $\boldsymbol{x}_c$.

We focus now on the computation of the term $\nabla_{\boldsymbol{x}_c}\boldsymbol{\theta}(\boldsymbol{x}_c)$. While for bilevel optimization problems in which the inner problem is not convex (e.g., when the learning algorithm is a neural network) this requires efficient numerical approximations [38], when the inner learning problem is convex, the gradient of interest can be computed in closed form. The underlying trick is to replace the inner learning problem (Eq. 3) with its Karush-Kuhn-Tucker (KKT) equilibrium conditions, i.e., $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{D}'_{\mathrm{tr}} \cup \mathcal{D}_p, \boldsymbol{\theta}) = \mathbf{0}$, and require such conditions to remain valid while updating $\boldsymbol{x}_c$ [5], [36], [38], [54]. To this end, we simply impose that their derivative with respect to $\boldsymbol{x}_c$ remains at equilibrium, i.e., $\nabla_{\boldsymbol{x}_c}\left(\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{D}'_{\mathrm{tr}} \cup \mathcal{D}_p, \boldsymbol{\theta})\right) = \mathbf{0}$. Now, it is clear that the function $\mathcal{L}$ depends explicitly on $\boldsymbol{x}_c$ in its first argument, and implicitly through the regression parameters $\boldsymbol{\theta}$. Thus, differentiating again with the chain rule, one yields the following linear system:

$$\nabla_{\boldsymbol{x}_c}\nabla_{\boldsymbol{\theta}}\mathcal{L} + \nabla_{\boldsymbol{x}_c}\boldsymbol{\theta}^\top \cdot \nabla_{\boldsymbol{\theta}}^2\mathcal{L} = \mathbf{0}. \qquad (5)$$

Finally, solving for $\nabla_{\boldsymbol{x}_c}\boldsymbol{\theta}$, one yields:

$$\nabla_{\boldsymbol{x}_c}\boldsymbol{\theta}^\top = \left[\begin{array}{cc} \frac{\partial \boldsymbol{w}}{\partial \boldsymbol{x}_c}^\top & \frac{\partial b}{\partial \boldsymbol{x}_c}^\top \end{array} \right] = -\nabla_{\boldsymbol{x}_c}\nabla_{\boldsymbol{\theta}}\mathcal{L}\left(\nabla_{\boldsymbol{\theta}}^2\mathcal{L}\right)^{-1}. \qquad (6)$$

For the specific form of $\mathcal{L}$ given in Eq. (1), it is not difficult to see that the aforementioned derivative becomes equal to

that reported in [54] (except for a factor of 2 arising from a different definition of the quadratic loss).

$$\nabla_{\boldsymbol{x}_c}\boldsymbol{\theta}^\top = -\frac{2}{n}\left[\begin{array}{cc} \mathbf{M} & \boldsymbol{w} \end{array}\right]\left[\begin{array}{cc} \boldsymbol{\Sigma}+\lambda\boldsymbol{g} & \boldsymbol{\mu} \\ \boldsymbol{\mu}^\top & 1 \end{array}\right]^{-1}, \quad (7)$$

where $\boldsymbol{\Sigma} = \frac{1}{n}\sum_i \boldsymbol{x}_i\boldsymbol{x}_i^\top$, $\mu = \frac{1}{n}\sum_i \boldsymbol{x}_i$, and $\mathbf{M} = \boldsymbol{x}_c\boldsymbol{w}^\top + (f(\boldsymbol{x}_c) - y_c)\,\mathbb{I}_d$. As in [54], the term $\mathbf{g}$ is zero for OLS and LASSO, the identity matrix $\mathbb{I}_d$ for ridge regression, and $(1-\rho)\mathbb{I}_d$ for the elastic net.

Objective Functions. In previous work, the main objective used for $\mathcal{W}$ has been typically a loss function computed on an untainted validation set $\mathcal{D}_{\mathrm{val}} = \{(\boldsymbol{x}_i', y_i')\}_{i=1}^m$ [5], [36], [38]. Notably, only Xiao et al. [54] have used a regularized loss function computed on the training data (excluding the poisoning points) as a proxy to estimate the generalization error on unseen data. The rationale was to avoid the attacker to collect an additional set of points. In our experiments, we consider both possibilities, always using the MSE as the loss function:

$$\mathcal{W}_{\mathrm{tr}}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^n\left(f(\boldsymbol{x}_i, \boldsymbol{\theta}) - y_i\right)^2 + \lambda\Omega(\boldsymbol{w}), \quad (8)$$

$$\mathcal{W}_{\mathrm{val}}(\mathcal{D}_{\mathrm{val}}, \boldsymbol{\theta}) = \frac{1}{m}\sum_{j=1}^m\left(f(\boldsymbol{x}_j', \boldsymbol{\theta}) - y_j'\right)^2. \quad (9)$$

The complete gradient $\nabla_{\boldsymbol{x}_c}\mathcal{W}$ (Eq. 4) for these two objectives can thus be computed by multiplying Eq. (7) respectively to:

$$\nabla_{\boldsymbol{\theta}}\mathcal{W}_{\mathrm{tr}} = \left[\begin{array}{c} \nabla_{\boldsymbol{w}}\mathcal{W}_{\mathrm{tr}} \\ \nabla_b\mathcal{W}_{\mathrm{tr}} \end{array}\right] \quad (10)$$

$$= \left[\begin{array}{c} \frac{2}{n}\sum_{i=1}^n(f(\boldsymbol{x}_i) - y_i)\boldsymbol{x}_i + \lambda\frac{\partial\Omega}{\partial\boldsymbol{w}}^\top \\ \frac{2}{n}\sum_{i=1}^n(f(\boldsymbol{x}_i) - y_i) \end{array}\right], \quad (11)$$

$$\nabla_{\boldsymbol{\theta}}\mathcal{W}_{\mathrm{val}} = \left[\begin{array}{c} \nabla_{\boldsymbol{w}}\mathcal{W}_{\mathrm{val}} \\ \nabla_b\mathcal{W}_{\mathrm{val}} \end{array}\right] = \left[\begin{array}{c} \frac{2}{m}\sum_{j=1}^m(f(\boldsymbol{x}_j) - y_j)\boldsymbol{x}_j \\ \frac{2}{m}\sum_{j=1}^m(f(\boldsymbol{x}_j) - y_j) \end{array}\right]. \quad (12)$$

Initialization strategies. We discuss here how to select the initial set $\mathcal{D}_p$ of poisoning points to be passed as input to the gradient-based optimization algorithm (Algorithm 1). Previous work on poisoning attacks has only dealt with classification problems [5], [36], [38], [54]. For this reason, the initialization strategy used in all previously-proposed approaches has been to randomly clone a subset of the training data and flip their labels. Dealing with regression opens up different avenues. We therefore consider two initialization strategies in this work. In both cases, we still select a set of points at random from the training set $\mathcal{D}_{\mathrm{tr}}$, but then we set the new response value y_c of each poisoning point in one of two ways: (i) setting $y_c = 1-y$, and (ii) setting $y_c = \mathrm{round}(1-y)$, where round rounds to the nearest 0 or 1 value (recall that the response variables are in $[0, 1]$). We call the first technique *Inverse Flipping* (InvFlip) and the second *Boundary Flipping* (BFlip). Worth remarking, we experimented with many techniques for selecting the feature values before running gradient descent, and found that surprisingly they do not have significant improvement over a simple uniform random choice. We thus report results only for the two aforementioned initialization strategies.

Baseline Gradient Descent (BGD) **Attack.** We are now in a position to define a baseline attack against which we will compare our improved attacks. In particular, as no poisoning attack has ever been considered in regression settings, we define as the baseline poisoning attack an adaptation from the attack by Xiao et al. [54]. In particular, as in Xiao et al. [54], we select $\mathcal{W}_{\mathrm{tr}}$ as the outer objective. To simulate label flips in the context of regression, we initialize the response variables of the poisoning points with the InvFlip strategy. We nevertheless test all the remaining three combinations of initialization strategies and outer objectives in our experiments.

Response Variable Optimization. This work is the first to consider poisoning attacks in regression settings. Within this context, it is worth remarking that response variables take on continuous values rather than categorical ones. Based on this observation, we propose here the first poisoning attack that jointly optimizes the feature values $\boldsymbol{x}_c$ of poisoning attacks *and* their associated response variable y_c. To this end, we extend the previous gradient-based attack by considering the optimization of $\boldsymbol{z}_c = (\boldsymbol{x}_c, y_c)$ instead of only considering $\boldsymbol{x}_c$. This means that all previous equations remain valid provided that we substitute $\nabla_{\boldsymbol{z}_c}$ to $\nabla_{\boldsymbol{x}_c}$. This clearly requires expanding $\nabla_{\boldsymbol{x}_c}\boldsymbol{\theta}$ by also considering derivatives with respect to y_c:

$$\nabla_{\boldsymbol{z}_c}\boldsymbol{\theta} = \left[\begin{array}{cc} \frac{\partial\boldsymbol{w}}{\partial\boldsymbol{x}_c} & \frac{\partial\boldsymbol{w}}{\partial y_c} \\ \frac{\partial b}{\partial\boldsymbol{x}_c} & \frac{\partial b}{\partial y_c} \end{array}\right], \quad (13)$$

and, accordingly, modify Eq. (7) as

$$\nabla_{\boldsymbol{z}_c}\boldsymbol{\theta}^\top = -\frac{2}{n}\left[\begin{array}{cc} \mathbf{M} & \boldsymbol{w} \\ -\boldsymbol{x}_c^\top & -1 \end{array}\right]\left[\begin{array}{cc} \boldsymbol{\Sigma}+\lambda\boldsymbol{g} & \boldsymbol{\mu} \\ \boldsymbol{\mu}^\top & 1 \end{array}\right]^{-1}. \quad (14)$$

The derivatives given in Eqs. (10)-(12) remain clearly unchanged, and can be pre-multiplied by Eq. (14) to obtain the final gradient $\nabla_{\boldsymbol{z}_c}\mathcal{W}$. Algorithm 1 can still be used to implement this attack, provided that both $\boldsymbol{x}_c$ and y_c are updated along the gradient $\nabla_{\boldsymbol{z}_c}\mathcal{W}$ (cf. Algorithm 1, line 7).

Theoretical Insights. We discuss here some theoretical insights on the bilevel optimization of Eqs. (2)-(3), which will help us to derive the basis behind the statistical attack introduced in the next section. To this end, let us first consider as the outer objective a non-regularized version of $\mathcal{W}_{\mathrm{tr}}$, which can be obtained by setting $\lambda = 0$ in Eq. (8). As we will see, in this case it is possible to compute simplified closed forms for the required gradients. Let us further consider another objective denoted with $\mathcal{W}_{\mathrm{tr}}'$, which, instead of optimizing the loss, optimizes the difference in predictions from the original, unpoisoned model $\boldsymbol{\theta}'$:

$$\mathcal{W}_{\mathrm{tr}}' = \frac{1}{n}\sum_{i=1}^n(f(\boldsymbol{x}_i, \boldsymbol{\theta}) - f(\boldsymbol{x}_i, \boldsymbol{\theta}'))^2.$$

In Appendix A, we show that $\mathcal{W}\mathrm{tr}$ and $\mathcal{W}_{\mathrm{tr}}'$ are interchangeable for our bilevel optimization problem. In particular, differentiating $\mathcal{W}_{\mathrm{tr}}'$ with respect to $\boldsymbol{z}_c = (\boldsymbol{x}_c, y_c)$ gives:

$$\frac{\partial\mathcal{W}_{\mathrm{tr}}'}{\partial\boldsymbol{x}_c} = \frac{2}{n}(f(\boldsymbol{x}_c, \boldsymbol{\theta}) - f(\boldsymbol{x}_c, \boldsymbol{\theta}'))(\boldsymbol{w}_0 - 2\boldsymbol{w})^\top \quad (15)$$

$$\frac{\partial\mathcal{W}_{\mathrm{tr}}'}{\partial y_c} = \frac{2}{n}(f(\boldsymbol{x}_c, \boldsymbol{\theta}) - f(\boldsymbol{x}_c, \boldsymbol{\theta}')). \quad (16)$$

The update rules defined by these gradients have nice interpretation. We see that $\frac{\partial \mathcal{W}_{\text{tr}}'}{\partial y_c}$ will update y_c to be further away from the original line than it was in the previous iteration. This is intuitive, as a higher distance from the line will push the line further in that direction. The update for $\boldsymbol{x}_c$ is slightly more difficult to understand, but by separating $(\boldsymbol{w}_0 - 2\boldsymbol{w})$ into $(-\boldsymbol{w}) + (\boldsymbol{w}_0 - \boldsymbol{w})$, we see that the $\boldsymbol{x}_c$ value is being updated in two directions summed together. The first is perpendicularly away from the regression line (like the y_c update step, the poison point should be as far as possible from the regression line). The second is parallel to the difference between the original regression line and the poisoned regression line (it should keep pushing in the direction it has been going). This gives us an intuition for how the poisoning points are being updated, and what an optimal poisoning point looks like.

B. Statistical-based Poisoning Attack (StatP)

Motivated by the aforementioned theoretical insights, we design a fast statistical attack that produces poisoned points with similar distribution as the training data. In StatP, we simply sample from a multivariate normal distribution with the mean and covariance estimated from the training data. Once we have generated these points, we round the feature values to the corners, exploiting the observation that the most effective poisoning points are near corners. Finally, we select the response variable's value at the boundary (either 0 or 1) to maximize the loss.

Note that, importantly, the StatP attack requires only black-box access to the model, as it needs to query the model to find the response variable (before performing the boundary rounding). It also needs minimal information to be able to sample points from the training set distribution. In particular, StatP requires an estimate of the mean and co-variance of the training data. However, StatP is agnostic to the exact regression algorithm, its parameters, and training set. Thus, it requires much less information on the training process than the optimization-based attacks. It is significantly faster than optimization-based attacks, though slightly less effective.

IV. DEFENSE ALGORITHMS

In this section, we describe existing defense proposals against poisoning attacks, and explain why they may not be effective under adversarial corruption in the training data. Then we present a new approach called TRIM, specifically designed to increase robustness against a range of poisoning attacks.

A. Existing defense proposals

Existing defense proposals can be classified into two categories: noise-resilient regression algorithms and adversarially-resilient defenses. We discuss these approaches below.

Noise-resilient regression. Robust regression has been extensively studied in statistics as a method to provide resilience against noise and outliers [26], [27], [51], [55]. The main idea behind these approaches is to identify and remove *outliers* from a dataset. For example, Huber [26] uses an outlier-robust loss function. RANSAC [17] iteratively trains a model to fit a
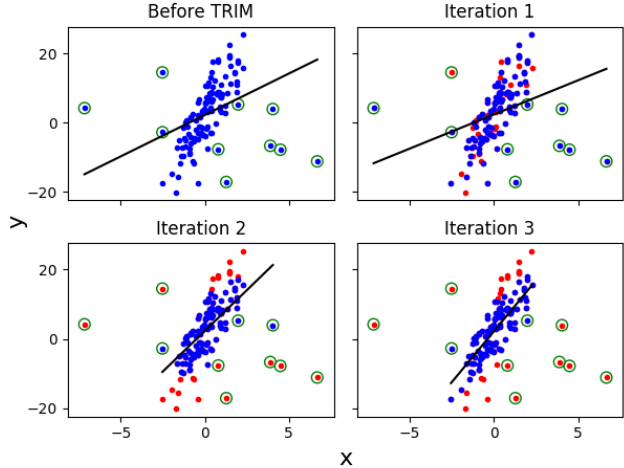

Fig. 2: Several iterations of the TRIM algorithm. Initial poisoned data is in blue in top left graph. The top right graph shows in red the initial randomly selected points removed from the optimization objective. In the following two iterations (bottom left and right graphs) the set of high-residual points is refined and the model becomes more robust.

subset of samples selected at random, and identifies a training sample as an outlier if the error when fitting the model to the sample is higher than a threshold.

While these methods provide robustness guarantees against noise and outliers, an adversary can still generate poisoning data that affects the trained model. In particular, an attacker can generate poisoning points that are very similar to the true data distribution (these are called *inliers*), but can still mislead the model. Our new attacks discussed in Section III generate poisoning data points which are akin to the pristine ones. For example, in StatP the poisoned points are chosen from a distribution that is similar to that of the training data (has the same mean and co-variance). It turns out that these existing regression methods are not robust against inlier attack points chosen to maximally mislead the estimated regression model.

Adversarially-resilient regression. Previously proposed adversarially-resilient regression algorithms typically provide guarantees under strong assumptions about data and noise distribution. For instance, Chen et al. [10], [11] assume that the feature set matrix satisfies $X^T X = I$ and data has sub-Gaussian distribution. Feng et al. [16] assume that the data and noise satisfy the sub-Gaussian assumption. Liu et al. [33] design robust linear regression algorithms robust under the assumption that the feature matrix has low rank and can be projected to a lower dimensional space. All these methods have provable robustness guarantees, but the assumptions on which they rely are not usually satisfied in practice.

B. TRIM *algorithm*

In this section, we propose a novel defense algorithm called TRIM with the goal of training a regression model with poisoned data. At an intuitive level, rather than simply removing outliers from the training set, TRIM takes a

Algorithm 2 [TRIM algorithm]
1: **Input**: Training data $\mathcal{D} = \mathcal{D}_{tr} \cup \mathcal{D}_p$ with $|\mathcal{D}| = N$; number of attack points $p = \alpha \cdot n$.
2: **Output**: $\boldsymbol{\theta}$.
3: $\mathcal{I}^{(0)} \leftarrow$ a random subset with size n of $\{1, ..., N\}$
4: $\boldsymbol{\theta}^{(0)} \leftarrow \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{I}^{(0)}, \boldsymbol{\theta})$ /* Initial estimation of $\boldsymbol{\theta}$*/
5: $i \leftarrow 0$ /* Iteration count */
6: **repeat**
7:     $i \leftarrow i + 1$;
8:     $\mathcal{I}^{(i)} \leftarrow$ subset of size n that min. $\mathcal{L}(\mathcal{D}^{\mathcal{I}^{(i)}}, \boldsymbol{\theta}^{(i-1)})$
9:     $\boldsymbol{\theta}^{(i)} \leftarrow \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}^{\mathcal{I}^{(i)}}, \boldsymbol{\theta})$ /* Current estimator */
10:    $R^{(i)} = \mathcal{L}(\mathcal{D}^{\mathcal{I}^{(i)}}, \boldsymbol{\theta}^{(i)})$ /* Current loss */
11: **until** $i > 1 \wedge R^{(i)} = R^{(i-1)}$ /* Convergence condition*/
12: **return** $\boldsymbol{\theta}^{(i)}$ /* Final estimator */.

principled approach. TRIM iteratively estimates the regression parameters, while at the same time training on a subset of points with lowest residuals in each iteration. In essence, TRIM uses a trimmed loss function computed on a different subset of residuals in each iteration. Our method is inspired by techniques from robust statistics that use trimmed versions of the loss function for robustness. Our main contribution is to apply trimmed optimization techniques for regularized linear regression in adversarial settings, and demonstrate their effectiveness compared to other defenses on a range of models and real-world datasets.

As in Section II, assume that the original training set is $\mathcal{D}_{tr}$ of size n, the attacker injects $p = \alpha \cdot n$ poisoned samples $\mathcal{D}_p$, and the poisoned training set $\mathcal{D} = \mathcal{D}_{tr} \cup \mathcal{D}_p$ is of size $N = (1 + \alpha)n$. We require that $\alpha < 1$ to ensure that the majority of training data is pristine (unpoisoned).

Our main observation is the following: we can train a linear regression model only using a subset of training points of size n. In the ideal case, we would like to identify all p poisoning points and train the regression model based on the remaining n legitimate points. However, the true distribution of the legitimate training data is clearly unknown, and it is thus difficult to separate legitimate and attack points precisely. To alleviate this, our proposed defense tries to identify a set of training points with lowest residuals relative to the regression model (these might include attack points as well, but only those that are "close" to the legitimate points and do not contribute much to poisoning the model). In essence, our TRIM algorithm provides a solution to the following optimization problem:

$$\min_{\boldsymbol{\theta}, \mathcal{I}} \mathcal{L}(\mathcal{D}^{\mathcal{I}}, \boldsymbol{\theta}) \quad \text{s.t. } I \subset [1, \ldots, N] \wedge |\mathcal{I}| = n. \quad (17)$$

We use the notation $\mathcal{D}^{\mathcal{I}}$ to indicate the data samples $\{(\boldsymbol{x}_i, y_i) \in \mathcal{D}\}_{i \in \mathcal{I}}$. Thus, we optimize the parameter $\boldsymbol{\theta}$ of the regression model and the subset I of points with smallest residuals at the same time. It turns out though that solving this optimization problem efficiently is quite challenging. A simple algorithm that enumerates all subsets I of size n of the training set is computationally inefficient. On the other

hand, if the true model parameters $\boldsymbol{\theta} = (\boldsymbol{w}, b)$ were known, then we could simply select points in set I that have lowest residual relative to $\boldsymbol{\theta}$. However, what makes this optimization problem difficult to solve is the fact that $\boldsymbol{\theta}$ is not known, and we do not make any assumptions on the true data distribution or the attack points.

To address these issues, our TRIM algorithm learns parameter $\boldsymbol{\theta}$ and distinguishes points with lowest residuals from training set alternately. We employ an iterative algorithm inspired by techniques such as alternating minimization or expectation maximization [13]. At the beginning of iteration i, we have an estimate of parameter $\boldsymbol{\theta}^{(i)}$. We use this estimate as a discriminator to identify all inliers, whose residual values are the n smallest ones. We do not consider points with large residuals (as they increase MSE), but use only the inliers to estimate a new parameter $\boldsymbol{\theta}^{(i+1)}$. This process terminates when the estimation converges and the loss function reaches a minimum. The detailed algorithm is presented in Algorithm 2. A graphical representation of three iterations of our algorithm is given in Figure 2. As observed in the figure, the algorithm iteratively finds the direction of the regression model that fits the true data distribution, and identifies points that are outliers.

We provide provable guarantees on the convergence of Algorithm 2 and the estimation accuracy of the regression model it outputs. First, Algorithm 2 is guaranteed to converge and thus it terminates in finite number of iterations, as stated in the following theorem.

Theorem 1. *Algorithm 2 terminates in a finite number of iterations.*

We do not explicitly provide a bound on the number of iterations needed for convergence, but it is always upper bounded by $\binom{N}{n}$. However, our empirical evaluation demonstrates that Algorithm 2 converges within few dozens of iterations at most.

We are next interested in analyzing the quality of the estimated model computed from Algorithm 2 (*adversarial world*) and how it relates to the pristine data (*ideal world*). However, relating these two models directly is challenging due to the iterative minimization used by Algorithm 2. We overcome this by observing that Algorithm 2 finds a *local minimum* to the optimization problem from (17). There is no efficient algorithm for solving (17) that guarantees the solution to be the global minimum of the optimization problem.

It turns out that we can provide a guarantee about the *global minimum* $\hat{\boldsymbol{\theta}}$ of (17) on poisoned data (under worst-case adversaries) in relation to the parameter $\boldsymbol{\theta}^{\star}$ learned by the original model on pristine data. In particular, Theorem 2 shows that $\hat{\boldsymbol{\theta}}$ "fits well" to at least $(1 - \alpha) \cdot n$ pristine data samples. Notably, it does not require any assumptions on how poisoned data is generated, thus it provides guarantees under worst-case adversaries.

Theorem 2. *Let $\mathcal{D}_{tr}$ denote the original training data, $\hat{\boldsymbol{\theta}}$ the global optimum for (17), and $\boldsymbol{\theta}^{\star} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}_{tr}, \boldsymbol{\theta})$ the estimator in the ideal world on pristine data. Assuming $\alpha < 1$, there exist a subset $\mathcal{D}' \subset \mathcal{D}_{tr}$ of $(1-\alpha) \cdot n$ pristine data samples*

such that

$$\mathrm{MSE}(\mathcal{D}', \hat{\boldsymbol{\theta}}) \leq \left(1 + \frac{\alpha}{1-\alpha}\right) \mathcal{L}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta}^\star). \qquad (18)$$

Note that the above theorem is stated without any assumptions on the training data distribution. This is one of the main difference from prior work [11], [16], which assume the knowledge of the mean and covariance of the legitimate data. In practice, such information on training data is typically unavailable. Moreover, an adaptive attacker can also inject poisoning samples to modify the mean and covariance of training data. Thus, our results are stronger than prior work in relying on fewer assumptions.

We now give an intuitive explanation about the above theorem, especially inequality (18). Since $\mathcal{D}_{\mathrm{tr}}$ is assumed to be the pristine dataset, and $\mathcal{D}'$ is a subset of $\mathcal{D}_{\mathrm{tr}}$ of size $(1-\alpha)n$, we know all data in $\mathcal{D}'$ is also pristine (not corrupted by the adversary). Therefore, the stationary assumption on pristine data distribution, which underpins all machine learning algorithms, guarantees that $\mathrm{MSE}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta})$ is close to $\mathrm{MSE}(\mathcal{D}', \boldsymbol{\theta})$ regardless of the choices of $\boldsymbol{\theta}$ and $\mathcal{D}'$, as long as α is small enough.

Next, we explain the left-hand side of inequality (18). This is the MSE of a subset of pristine samples $\mathcal{D}'$ using $\hat{\boldsymbol{\theta}}$ computed by the TRIM algorithm in the adversarial world. Based on the discussion above, the left-hand side is close to the MSE of the pristine data $\mathcal{D}_{\mathrm{tr}}$ using the adversarially learned estimator $\hat{\boldsymbol{\theta}}$. Thus, inequality (18) essentially provides an upper bound on the worst-case MSE using the estimator $\hat{\boldsymbol{\theta}}$ output by Algorithm 2 from the poisoned data.

To understand what upper bound Theorem 2 guarantees, we need to understand the right-hand side of inequality (18). We use OLS regression (without regularization) as an example to explain the intuition of the right-hand side. In OLS we have $\mathcal{L}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta}^\star) = \mathrm{MSE}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta}^\star)$, which is the MSE using the "best-case" estimator computed in the ideal world. Therefore, the right-hand side of inequality (18) is proportional to the ideal world MSE, with a factor of $(1+\frac{\alpha}{1-\alpha})$. When $\alpha \leq 20\%$, we notice that this factor is at most $1.25\times$.

Therefore, informally, Theorem 2 essentially guarantees that, the ratio of the worst-case MSE by solving (18) computed in the adversarial world over best-case MSE computed in ideal world for a linear model is at most 1.25. Note that since Algorithm 2 may not always find the global minimum of (17), we empirically examine this ratio of the worst-case to best-case MSEs. Our empirical evaluation shows that in most of our experiments, this ratio for TRIM is less than $1.01\times$, which is much smaller than all existing defenses.

For other models whose loss function includes the regularizer term (Lasso, ridge, and elastic net), the right-hand side of (18) includes the same term as well. This may allow the blowup of the worst-case MSE in the adversarial world with respect to the best-case MSE to be larger; however, we are not aware of any technique to trigger this worst-case scenario, and our empirical evaluation shows that the blowup is typically less than 1% as mentioned above.

The proofs of Theorem 1 and 2 can be found in Appendix B.

V. Experimental evaluation

We implemented our attack and defense algorithms in Python, using the numpy and sklearn packages. Our code is available at https://github.com/jagielski/manip-ml. We ran our experiments on four 32 core Intel(R) Xeon(R) CPU E5-2440 v2 @ 1.90GHz machines. We parallelize our optimization-based attack implementations to take advantage of the multi-core capabilities. We use the standard cross-validation method to split the datasets into 1/3 for training, 1/3 for testing, and 1/3 for validation, and report results as averages over 5 runs. We use two main metrics for evaluating our algorithms: MSE for the effectiveness of the attacks and defenses, and running time for their cost.

We describe the datasets we used for our experiments in Section V-A. We then systematically analyze the performance of the new attacks and compare them against the baseline attack algorithm in Section V-B. Finally, we present the results of our new TRIM algorithm and compare it with previous methods from robust statistics in Section V-C.

A. Datasets

We used three public regression datasets in our experimental evaluation. We present some details and statistics about each of them below.

Health care dataset. This dataset includes 5700 patients, where the goal is to predict the dosage of anticoagulant drug Warfarin using demographic information, indication for Warfarin use, individual VKORC1 and CYP2C9 genotypic data, and use of other medications affected by related VKORC1 and CYP2C9 polymorphisms [45]. As is standard practice for studies using this dataset (see [19]), we only select patients with INR values between 2 and 3. The INR is a ratio that represents the amount of time it takes for blood to clot, with a therapeutic range of 2-3 for most patients taking Warfarin. The dataset includes 67 features, resulting in 167 features after one-hot encoding categorical features and normalizing numerical features as above.

Loan dataset. This dataset contains information regarding loans made on the Lending Club peer-to-peer lending platform [29]. The predictor variables describe the loan attributes, including information such as total loan size, interest rate, and amount of principal paid off, as well as the borrower's information, such as number of lines of credit, and state of residence. The response variable is the interest rate of a loan. Categorical features, such as the purpose of the loan, are one-hot encoded, and numerical features are normalized into [0,1]. The dataset contains 887,383 loans, with 75 features before pre-processing, and 89 after. Due to its large scale, we sampled a set of 5000 records for our poisoning attacks.

House pricing dataset. This dataset is used to predict house sale prices as a function of predictor variables such as square footage, number of rooms, and location [28]. In total, it

Model	Dataset	Init	Argument	Objective
Ridge	Health	BFlip	(x,y)	$\mathcal{W}_{\mathrm{tr}}$
	Loan	BFlip	x	$\mathcal{W}_{\mathrm{val}}$
	House	BFlip	(x,y)	$\mathcal{W}_{\mathrm{tr}}$
LASSO	Health	BFlip	(x,y)	$\mathcal{W}_{\mathrm{tr}}$
	Loan	BFlip	(x,y)	$\mathcal{W}_{\mathrm{val}}$
	House	InvFlip	(x,y)	$\mathcal{W}_{\mathrm{val}}$

TABLE I: Best performing optimization attack OptP for Ridge and LASSO regression.

Init	Argument	Objective	Poisoning rates		
			12%	16%	20%
InvFlip	x	$\mathcal{W}_{\mathrm{tr}}$	0.026	0.027	0.027
BFlip	x	$\mathcal{W}_{\mathrm{tr}}$	0.028	0.032	0.033
InvFlip	(x,y)	$\mathcal{W}_{\mathrm{tr}}$	0.026	0.027	0.029
BFlip	(x,y)	$\mathcal{W}_{\mathrm{tr}}$	0.029	0.0316	0.032
BFlip	(x,y)	$\mathcal{W}_{\mathrm{val}}$	0.030	0.0338	0.0376

TABLE II: MSEs of optimization attacks for LASSO on loan data. BGD is the first row.

Init	Argument	Objective	Poisoning rates		
			12%	16%	20%
InvFlip	x	$\mathcal{W}_{\mathrm{tr}}$	0.034	0.047	0.054
BFlip	x	$\mathcal{W}_{\mathrm{tr}}$	0.08	0.145	0.172
InvFlip	(x,y)	$\mathcal{W}_{\mathrm{tr}}$	0.04	0.047	0.052
InvFlip	(x,y)	$\mathcal{W}_{\mathrm{val}}$	0.369	0.369	0.369
BFlip	(x,y)	$\mathcal{W}_{\mathrm{tr}}$	0.08	0.145	0.172

TABLE III: MSEs of optimization attacks for LASSO on house data. BGD is the first row.

includes 1460 houses and 81 features. We preprocess by one-hot encoding all categorical features and normalize numerical features, resulting in 275 total features.

B. New poisoning attacks

In this section, we perform experiments on the three regression datasets (health care, loan, and house pricing) to evaluate the newly proposed attacks, and compare them against the baseline BGD [54] for four regression models. For each dataset we select a subset of 1400 records (this is the size of the house dataset, and we wanted to use the same number of records for all datasets). We use MSE as the metric for assessing the effectiveness of an attack, and also measure the attacks' running times. We vary the poisoning rate between 4% and 20% at intervals of 4% with the goal of inferring the trend in attack success. More details about hyperparameter setting are presented in Appendix C.

Figures 3 and 4 show the MSE of each attack for ridge and LASSO regression. We picked these two models as they are the most popular linear regression models. We plot the baseline attack BGD, statistical attack StatP, as well as our best performing optimization attack (called OptP). Details on OptP are given in Table I. Additional results for the Contagio PDF classification dataset are given in Appendix C.

Below we pose several research questions to elucidate the benefits, and in some cases limitations, of these attacks.

1) Question 1: Which optimization strategies are most effective for poisoning regression?: Our results confirm that the optimization framework we design is effective at poisoning different models and datasets. Our new optimization attack OptP improves upon the baseline BGD attack by a factor of 6.83 in the best case. The OptP attack could achieve MSEs by a factor of 155.7 higher than the original models.

As discussed in Section III, our optimization framework has several instantiations, depending on: (1) The initialization strategy (InvFlip or BFlip); (2) The optimization variable (x or (x,y)); and (3) The objective of the optimization ($\mathcal{W}_{\mathrm{tr}}$ or $\mathcal{W}_{\mathrm{val}}$). For instance, BGD is given by (InvFlip, x, $\mathcal{W}_{\mathrm{tr}}$). We show that each of these dimensions has an important effect in generating successful attacks. Table I shows the best optimization attack for each model and dataset, while Tables II and III provide examples of different optimization attacks for LASSO on the loan and house datasets, respectively.

We highlight several interesting observations. First, boundary flip BFlip is the preferred initialization method, with only one case (LASSO regression on house dataset) in which InvFlip performs better in combination with optimizing (x,y)

under objective $\mathcal{W}_{\mathrm{val}}$. For instance, in LASSO on house dataset, BFlip alone can achieve a factor of 3.18 higher MSE than BGD using InvFlip. In some cases the optimization by y can achieve higher MSEs even starting with non-optimal y values as the gradient ascent procedure is very effective (see for example the attack (InvFlip, (x,y), $\mathcal{W}_{\mathrm{val}}$) in Table III). However, the combination of optimization by x with InvFlip initialization (as used by BGD) is outperformed in all cases by either BFlip or (x,y) optimization.

Second, using both (x,y) as optimization arguments is most effective compared to simply optimizing by x as in BGD. Due to the continuous response variables in regression, optimizing by y plays a large role in making the attacks more effective. For instance, optimizing by (x,y) with BFlip initialization and $\mathcal{W}_{\mathrm{val}}$ achieves a factor of 6.83 improvement in MSE compared to BGD on house dataset with LASSO regression.

Third, the choice of the optimization objective is equally important for each dataset and model. $\mathcal{W}_{\mathrm{val}}$ can improve over $\mathcal{W}_{\mathrm{tr}}$ by a factor of 7.09 (on house for LASSO), by 17.5% (on loan for LASSO), and by 30.4% (on loan for ridge) when the initialization points and optimization arguments are the same.

Thus, all three dimensions in our optimization framework are influential in improving the success of the attack. The optimal choices are dependent on the data distribution, such as feature types, sparsity of the data, ratio of records over data dimension, and data linearity. In particular, we noticed that for non-linear datasets (such as loan), the original MSE is already high before the attack and all the attacks that we tested perform worse than in cases when the legitimate data fits a linear model (i.e., it is close to the regression hyperplane). The reason may be that, in the latter case, poisoning samples may be shifted farther away from the legitimate data (i.e., from the regression hyperplane), and thus have a greater impact than in the former case, when the legitimate data is already more evenly and non-linearly distributed in feature space. Nevertheless, our attacks are able to successfully poison a range of models and datasets.

2) Question 2: How do optimization and statistical attacks compare in effectiveness and performance?: In general,

(a) Health Care Dataset
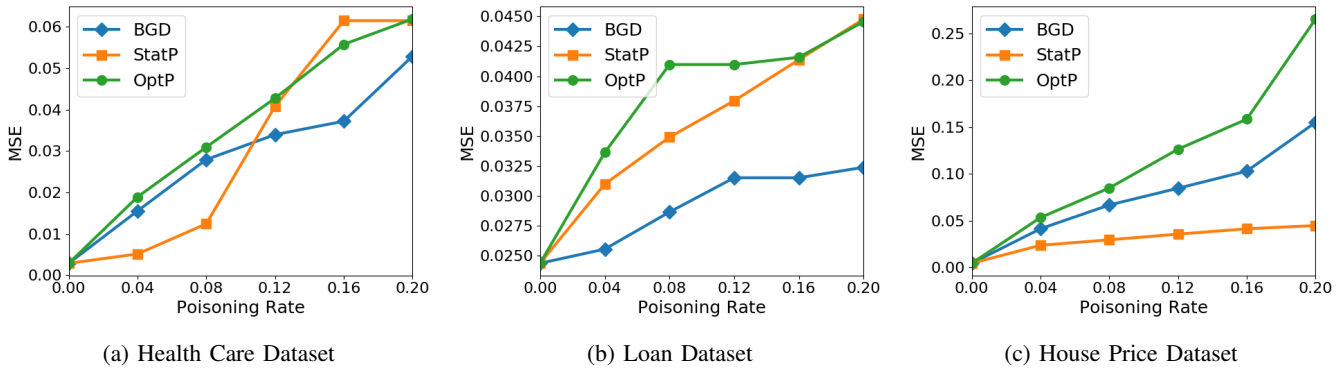
(b) Loan Dataset

(c) House Price Dataset

Fig. 3: MSE of attacks on ridge regression on the three datasets. Our new optimization (OptP) and statistical (StatP) attacks are more effective than the baseline. OptP is best optimization attack according to Table I.
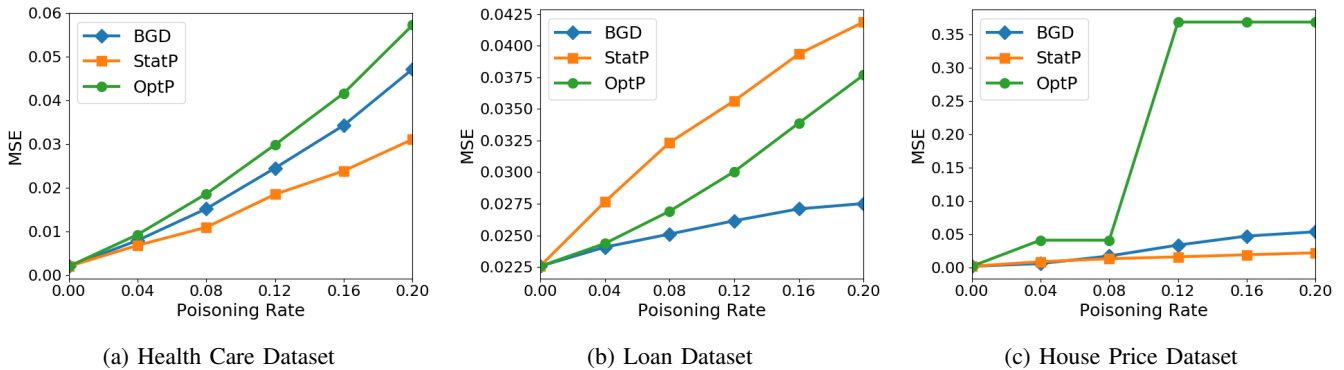

(a) Health Care Dataset

(b) Loan Dataset

(c) House Price Dataset

Fig. 4: MSE of attacks on LASSO on the three datasets. As for ridge, we find that StatP and OptP are able to poison the dataset very effectively, outperforming the baseline (BGD). OptP is best optimization attack according to Table I.

optimization-based attacks (BGD and OptP) outperform the statistical-based attack StatP in effectiveness. This is not surprising to us, as StatP uses much less information about the training process to determine the attack points. Interestingly, we have one case (LASSO regression on loan dataset) in which StatP outperforms the best optimization attack OptP by 11%. There are also two instances on ridge regression (health and loan datasets) in which StatP and OptP perform similarly. These cases show that StatP is a reasonable attack when the attacker has limited knowledge about the learning system.

The running time of optimization attacks is proportional to the number of iterations required for convergence. On the highest-dimensional dataset, house prices, we observe OptP taking about 337 seconds to complete for ridge and 408 seconds for LASSO. On the loan dataset, OptP finishes LASSO poisoning in 160 seconds on average. As expected, the statistical attack is extremely fast, with running times on the order of a tenth of a second on the house dataset and a hundredth of a second on the loan dataset to generate the same number of points as OptP. Therefore, our attacks exhibit clear tradeoffs between effectiveness and running times, with optimization attacks being more effective than statistical attacks, at the expense of higher computational overhead.

3) Question 3: What is the potential damage of poisoning in real applications?: We are interested in understanding the

effect of poisoning attacks in real applications, and perform a case study on the health-care dataset. Specifically, we translate the MSE results obtained with our attacks into application specific parameters. In the health care application, the goal is to predict medicine dosage for the anticoagulant drug Warfarin. In Table IV, we show first statistics on the medicine dosage predicted by the original regression models (without poisoning), and then the absolute difference in the amount of dosage prescribed after the OptP poisoning attack. We find that all linear regression models are vulnerable to poisoning, with 75% of patients having their dosage changed by 93.49%, and half of patients having their dosage changed by 139.31% on LASSO. For 10% of patients, the increase in MSE is devastating to a maximum of 359% achieved for LASSO regression. These results are for 20% poisoning rate, but it turns out that the attacks are also effective at smaller poisoning rates. For instance, at 8% poisoning rate, the change in dosage is 75.06% for half of patients.

Thus, the results demonstrate the effectiveness of our new poisoning attacks that induce significant changes to the dosage of most patients with a small percentage of poisoned points added by the attacker.

4) Question 4: What are the transferability properties of our attacks?: Our transferability analysis for poisoning attacks is based on the black-box scenario discussed in Sect. II, in

Quantile	Initial Dosage	Ridge Diff	LASSO Diff
0.1	15.5	31.54%	37.20%
0.25	21	87.50%	93.49%
0.5	30	150.99%	139.31%
0.75	41.53	274.18%	224.08%
0.9	52.5	459.63%	358.89%

TABLE IV: Initial dosage distribution (mg/wk) and percentage difference between original and predicted dosage after OptP attack at 20% poisoning rate (health care dataset).

Dataset	Attack	LASSO	Ridge
Health	OptP	1.092	1.301
	StatP	0.971	0.927
Loan	OptP	1.028	1.100
	StatP	1.110	0.989
House	OptP	1.779	0.479
	StatP	1.034	0.886

TABLE V: Transferability of OptP and StatP attacks. Presented are the ratio of the MSE obtained with transferred attacks over original attacks. Values below 1 represent original attacks outperforming transferred attacks, while values above 1 represent transferred attacks outperforming original attacks.

which the attacker uses a substitute training set $\mathcal{D}'_{\mathrm{tr}}$ to craft the poisoning samples, and then tests them against the targeted model (trained on $\mathcal{D}_{\mathrm{tr}}$). Our results, averaged on 5 runs, are detailed in Table V, which presents the ratio between transferred and original attacks. Note that the effectiveness of transferred attacks is very similar to that of the original attacks, with some outliers on the house dataset. For instance, the statistical attack StatP achieves transferred MSEs within 11.4% of the original ones. The transferred OptP attacks have lower MSEs by 3% than the original attack on LASSO. At the same time, transferred attacks could also improve the effectiveness of the original attacks: by 30% for ridge, and 78% for LASSO. We conclude that, interestingly, our most effective poisoning attacks (OptP and StatP) tend to have good transferability properties. There are some exceptions (ridge on house dataset), which deserve further investigation in future work. In most cases the MSEs obtained when using a different training set for both attacks is comparable to MSEs obtained when the attack is mounted on the actual training set.

Summary of poisoning attack results.

- We introduce a new optimization framework for poisoning regression, which manages to improve upon BGD by a factor of 6.83. The best OptP attack selects the initialization strategy, optimization argument, and optimization objective to achieve maximum MSEs.
- We find that our statistical-based attack (StatP) works reasonably well in poisoning all datasets and models, is efficient in running time, and needs minimal information on the model. Our optimization-based attack OptP takes longer to run, needs more information on the model, but can be more effective in poisoning than StatP if properly configured.
- In a health care case study, we find that our OptP attack can cause half of patients' Warfarin dosages to change by an average of 139.31%. One tenth of these patients can have their dosages changed by 359%, demonstrating the devastating consequences of poisoning.
- We find that both our statistical and optimization attacks have good transferability properties, and still perform well with minimal difference in accuracy, when applied to different training sets.

C. Defense algorithms

In this section, we evaluate our proposed TRIM defense and other existing defenses from the literature (Huber, RANSAC,

Chen, and RONI) against the best performing optimization attacks from the previous section (OptP). We test two well-known methods from robust statistics: Huber regression [26] and RANSAC [17], available as implementations in Python's sklearn package. Huber regression modifies the loss function from the standard MSE to reduce the impact of outliers. It does this by using quadratic terms in the loss function for points with small residuals and linear terms for points with large residuals. The threshold where linear terms start being used is tuned by a parameter $\epsilon > 1$, which we set by selecting the best of 5 different values: $\{1.1, 1.25, 1.35, 1.5, 2\}$. RANSAC builds a model on a random sample of the dataset, and computes the number of points that are outliers from that model. If there are too many outliers, the model is rejected and a new model is computed on a different random dataset sample. The size of the initial random sample is a parameter that requires tuning - we select 5 different values, linearly interpolating from 25 to the total number of clean data, and select the value which has the lowest MSE. If the number of outliers is smaller than the number of poisoning points, we retain the model.

We also compare against our own implementation of the robust regression method by Chen et al. [11] from the machine learning community, and the RONI method from the security community [39]. Chen picks the features of highest influence using an outlier resilient dot product computation. We vary the number of features selected by Chen (the only parameter in the algorithm) between 1 and 9 and pick the best results. We find that Chen has highly variable performance, having MSE increases of up to a factor of 63,087 over the no defense models, and we decided to not include it in our graphs. The poor performance of Chen is due to the strong assumptions of the technique (sub-Gaussian feature distribution and covariance matrix $\mathbf{X}^T\mathbf{X} = \mathbb{I}$.), that are not met by our real world datasets. While we were able to remove the assumption that all features had unit variance through robust scaling (using the robust dot product provided by their work), removing the covariance terms would require a robust matrix inversion, which we consider beyond the scope of our work.

RONI (Reject On Negative Impact) was proposed in the context of spam filters and attempts to identify outliers by observing the performance of a model trained with and without each point. If the performance degrades too much on a sampled validation set (which may itself contain outliers), the point is

(a) Health Care Dataset   (b) Loan Dataset   (c) House Price Dataset
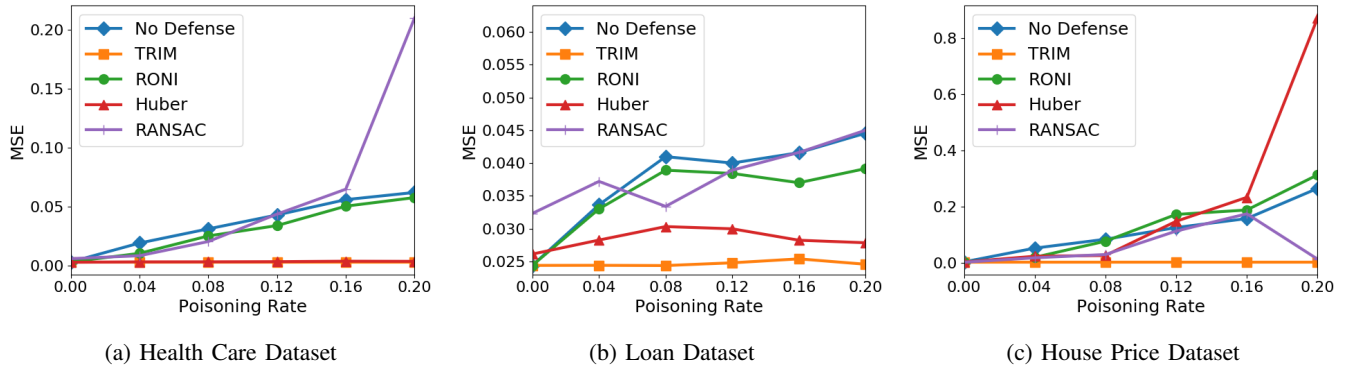
Fig. 5: MSE of defenses on ridge on the three datasets. We exclude Chen from the graphs due to its large variability. Defenses are evaluated against the OptP attack. The only defense that consistently performs well in these situations is our proposed TRIM defense, with RANSAC, Huber, and RONI actually performing worse than the undefended model in some cases.
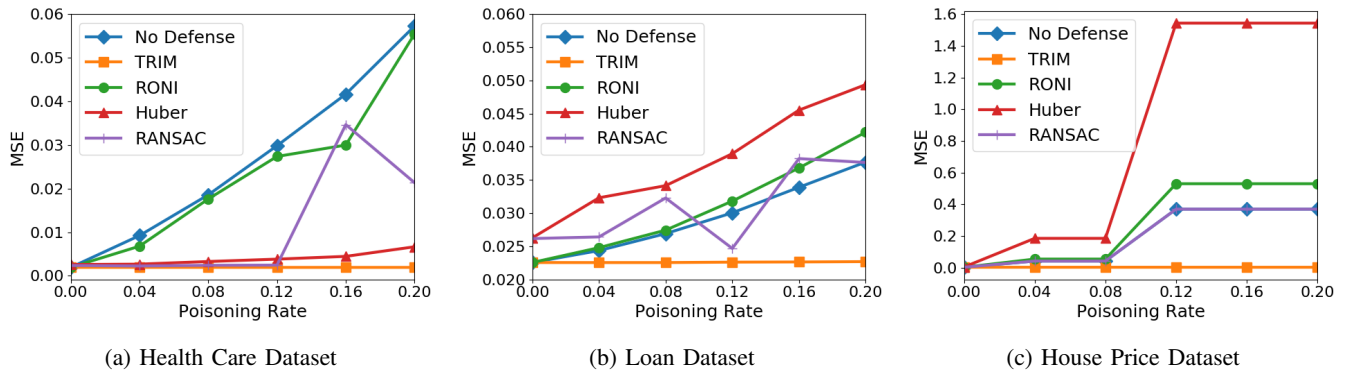

(a) Health Care Dataset   (b) Loan Dataset   (c) House Price Dataset

Fig. 6: MSE of defenses on LASSO. We exclude Chen from the graphs due to its large variability. Defenses are evaluated against the most effective attack OptP. As with ridge, the only defense that consistently performs well is our TRIM defense.

identified as an outlier and not included in the model. This method has some success in the spam scenario due to the ability of an adversary to send a single spam email with all words in dictionary, but is not applicable in other settings in which the impact of each point is small. We set the size of the validation set to 50, and pick the best points on average from 5 trials, as in the original paper. The size of the training dataset is selected from the same values as RANSAC's initial sample size.

We show in Figures 5 and 6 MSEs for ridge and LASSO regression for the original model (no defense), the TRIM algorithm, as well as the Huber, RANSAC, and RONI methods. We pose three research questions next:

1) Question 1: Are known methods effective at defending against poisoning attacks?: As seen in Figures 5 and 6, existing techniques (Huber regression, RANSAC, and RONI), are not consistently effective at defending against our presented attacks. For instance, for ridge models, the OptP attack increases MSE over unpoisoned models by a factor of 60.22 (on the house dataset). Rather than decreasing the MSE, Huber regression in fact increases the MSE over undefended ridge models by a factor of 3.28. RONI also increases the MSE of undefended models by 18.11%. RANSAC is able to reduce MSE, but it is still greater by a factor of 4.66 than that of the original model. The reason for this poor performance is that

robust statistics methods are designed to remove or reduce the effect of outliers from the data, while RONI can only identify outliers with high impact on the trained models. Our attacks generate inlier points that have similar distribution as the training data, making these previous defenses ineffective.

2) Question 2: What is the robustness of the new defense TRIM compared to known methods?: Our TRIM technique is much more effective at defending against all attacks than the existing techniques are. For ridge and LASSO regression, TRIM's MSE is within 1% of the original models in all cases. Interestingly, on the house price dataset the MSE of TRIM is lower by 6.42% compared to unpoisoned models for LASSO regression. TRIM achieves MSEs much lower than existing methods, improving Huber by a factor of 1295.45, RANSAC by a factor of 75, and RONI by a factor of 71.13. This demonstrates that the TRIM technique is a significant improvement over prior work at defending against these poisoning attacks.

3) Question 3: What is the performance of various defense algorithms?: All of the defenses we evaluated ran in a reasonable amount of time, but TRIM is the fastest. For example, on the house dataset, TRIM took an average of 0.02 seconds, RANSAC took an average of 0.33 seconds, Huber took an average of 7.86 seconds, RONI took an average of 15.69 seconds and Chen took an average of 0.83 seconds. On the health care dataset, TRIM took an average of 0.02 seconds,

RANSAC took an average of 0.30 seconds, Huber took an average of 0.37 seconds, RONI took an average of 14.80 seconds, and Chen took an average of 0.66 seconds. There is some variance depending on the dataset and the number of iterations to convergence, but TRIM is consistently faster than other methods.

Summary of defense results.

- We find that previous defenses (RANSAC, Huber, Chen, and RONI) do not work very well against our poisoning attacks. As seen in Figures 5-6, previous defenses can in some cases increase the MSEs over unpoisoned models.
- Our proposed defense, TRIM, works very well and significantly improves the MSEs compared to existing defenses. For all attacks, models, and datasets, the MSEs of TRIM are **within 1%** of the unpoisoned model MSEs. In some cases TRIM achieves lower MSEs than those of unpoisoned models (by 6.42%).
- All of the defenses we tested ran reasonably quickly. TRIM was the fastest, running in an average of 0.02 seconds on the house price dataset.

VI. RELATED WORK

The security of machine learning has received a lot of attention in different communities (e.g., [2], [4], [14], [25], [34]. Different types of attacks against machine learning algorithms have been designed and analyzed, including *evasion attacks* (e.g., [3], [8], [20], [42], [43], [49], [50]), and *privacy attacks* (e.g., [18], [19], [47]). In *poisoning attacks* the attacker manipulates or injects malicious data during training to cause either *availability attacks* (inducing an effect on the trained model) or *targeted attacks* (inducing an effect on specific data points) [5], [25], [36], [38], [54].

In the security community, practical poisoning attacks have been demonstrated in worm signature generation [41], [44], spam filters [39], network traffic analysis systems for detection of DoS attacks [46], sentiment analysis on social networks [40], crowdsourcing [53], and health-care [37]. In supervised learning settings, Newsome et al. [41] have proposed *red herring attacks* that add spurious words (features) to reduce the maliciousness score of an instance. These attacks work against conjunctive and Bayes learners for worm signature generation. Perdisci et al. [44] practically demonstrate how an attacker can inject noise in the form of suspicious flows to mislead worm signature classification. Nelson et al. [39] present both availability and targeted poisoning attacks against the public SpamBayes spam classifier. Venkataraman et al. [52] analyze the theoretical limits of poisoning attacks against signature generation algorithms by proving bounds on false positives and false negatives for certain adversarial capabilities.

In unsupervised settings, Rubinstein et al. [46] examined how an attacker can systematically inject traffic to mislead a PCA anomaly detection system for DoS attacks. Kloft and Laskov [31] demonstrated *boiling frog attacks* on centroid anomaly detection that involve incremental contamination of systems using retraining. Theoretical online centroid anomaly detection analysis has been discussed in [31]. Ciocarlie et al. [12] discuss sanitization methods against time-based anomaly detectors in which multiple micro-models are built and compared over time to identify poisoned data. The assumption in their system is that the attacker only controls data generated during a limited time window.

In the machine learning and statistics communities, earliest treatments consider the robustness of learning to noise, including the extension of the PAC model by Kearns and Li [30], as well as work on robust statistics [7], [27], [51], [55]. In adversarial settings, robust methods for dealing with arbitrary corruptions of data have been proposed in the context of linear regression [11], high-dimensional sparse regression [10], logistic regression [16], and linear regression with low rank feature matrix [33]. These methods are based on assumptions on training data such as sub-Gaussian distribution, independent features, and low-rank feature space. Biggio et al. [5] pioneered the research of optimizing poisoning attacks for kernel-based learning algorithms such as SVM. Similar techniques were later generalized to optimize data poisoning attacks for several other important learning algorithms, such as feature selection for classification [54], topic modeling [35], autoregressive models [1], collaborative filtering [32], and simple neural network architectures [38].

VII. CONCLUSIONS

We perform the first systematic study on poisoning attacks and their countermeasures for linear regression models. We propose a new optimization framework for poisoning attacks and a fast statistical attack that requires minimal knowledge of the training process. We also take a principled approach in designing a new robust defense algorithm that largely outperforms existing robust regression methods. We extensively evaluate our proposed attack and defense algorithms on several datasets from health care, loan assessment, and real estate domains. We demonstrate the real implications of poisoning attacks in a case study health application. We finally believe that our work will inspire future research towards developing more secure learning algorithms against poisoning attacks.

REFERENCES

[1] S. Alfeld, X. Zhu, and P. Barford. Data poisoning attacks against autoregressive models. In *AAAI*, 2016.

[2] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25. ACM, 2006.

[3] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, editors, *Machine Learning and Knowledge Discovery in Databases (ECML PKDD), Part III*, volume 8190 of *LNCS*, pages 387–402. Springer Berlin Heidelberg, 2013.

[4] B. Biggio, G. Fumera, and F. Roli. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):984–996, April 2014.

[5] B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. In *ICML*, 2012.

[6] B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *ArXiv e-prints*, 2018.

[7] E. J. Candes, X. Li, Y. Ma, and J. Wright. Robust principal component analysis. *Journal of the ACM*, 58(3), 2011.

[8] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *Proc. IEEE Security and Privacy Symposium*, S&P, 2017.

[9] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning systems using data poisoning. *ArXiv e-prints*, abs/1712.05526, 2017.

[10] Y. Chen, C. Caramanis, and S. Mannor. Robust high dimensional sparse regression and matching pursuit. arXiv:1301.2725, 2013.

[11] Y. Chen, C. Caramanis, and S. Mannor. Robust sparse regression under adversarial corruption. In *Proc. International Conference on Machine Learning*, ICML, 2013.

[12] G. F. Cretu-Ciocarlie, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *Proc. IEEE Security and Privacy Symposium*, S&P, 2008.

[13] I. Csiszar and G. Tusnady. Information geometry and alternating minimization procedures. *Statistics and Decisions*, 1:205–237, 1984.

[14] N. Dalvi, P. Domingos, S. Sanghai, D. Verma, et al. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108. ACM, 2004.

[15] D. Faggella. Machine learning healthcare applications – 2017 and beyond. https://www.techemergence.com/machine-learning-healthcare-applications/, 2016.

[16] J. Feng, H. Xu, S. Mannor, , and S. Yan. Robust logistic regression and classification. In *Advances in Neural Information Processing Systems*, NIPS, 2014.

[17] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[18] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, CCS, 2015.

[19] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *USENIX Security*, pages 17–32, 2014.

[20] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. arXiv:1412.6572, 2014.

[21] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. In *NIPS Workshop on Machine Learning and Computer Security*, volume abs/1708.06733, 2017.

[22] S. Hao, A. Kantchelian, B. Miller, V. Paxson, and N. Feamster. PREDATOR: Proactive recognition and elimination of domain abuse at time-of-registration. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security*, CCS, 2016.

[23] P. Harsha. Senate committee examines the "dawn of artificial intelligence". Computing Research Policy Blog. http://cra.org/govaffairs/blog/2016/11/senate-committee-examines-dawn-artificial-intelligence/, 2016.

[24] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[25] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58. ACM, 2011.

[26] P. J. Huber. Robust estimation of a location parameter. *Annals of Statistics*, 53(1):73–101, 1964.

[27] P. J. Huber. *Robust statistics*. Springer, 2011.

[28] Kaggle. House Prices: Advanced Regression Techniques. https://www.kaggle.com/c/house-prices-advanced-regression-techniques. Online; accessed 8 May 2017.

[29] W. Kan. Lending Club Loan Data. https://www.kaggle.com/wendykan/lending-club-loan-data, 2013. Online; accessed 8 May 2017.

[30] M. Kearns and M. Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837, 1993.

[31] M. Kloft and P. Laskov. Security analysis of online centroid anomaly detection. *The Journal of Machine Learning Research*, 13(1):3681–3724, 2012.

[32] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. In *Advances In Neural Information Processing Systems*, pages 1885–1893, 2016.

[33] C. Liu, B. Li, Y. Vorobeychik, and A. Oprea. Robust linear regression against training data poisoning. In *Proc. Workshop on Artificial Intelligence and Security*, AISec, 2017.

[34] D. Lowd and C. Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 641–647. ACM, 2005.

[35] S. Mei and X. Zhu. The security of latent dirichlet allocation. In *AISTATS*, 2015.

[36] S. Mei and X. Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *29th AAAI Conf. Artificial Intelligence (AAAI '15)*, 2015.

[37] M. Mozaffari Kermani, S. Sur-Kolay, A. Raghunathan, and N. K. Jha. Systematic poisoning attacks on and defenses for machine learning in healthcare. *IEEE Journal of Biomedical and Health Informatics*, 19(6):1893–1905, 2014.

[38] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In B. M. Thuraisingham, B. Biggio, D. M. Freeman, B. Miller, and A. Sinha, editors, *10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pages 27–38, New York, NY, USA, 2017. ACM.

[39] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. Sutton, J. Tygar, and K. Xia. Exploiting machine learning to subvert your spam filter. In *Proc. First USENIX Workshop on Large-Scale Exploits and Emergent Threats*, LEET, 2008.

[40] A. Newell, R. Potharaju, L. Xiang, and C. Nita-Rotaru. On the practicality of integrity attacks on document-level sentiment analysis. In *Proc. Workshop on Artificial Intelligence and Security*, AISec, 2014.

[41] J. Newsome, B. Karp, and D. Song. Paragraph: Thwarting signature learning by training maliciously. In *Recent advances in intrusion detection*, pages 81–105. Springer, 2006.

[42] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *Proc. IEEE European Security and Privacy Symposium*, Euro S&P, 2017.

[43] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proc. IEEE Security and Privacy Symposium*, S&P, 2016.

[44] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *Proc. IEEE Security and Privacy Symposium*, S&P, 2006.

[45] PharmGKB. Downloads - IWPC Data. https://www.pharmgkb.org/downloads/, 2014. Online; accessed 8 May 2017.

[46] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S. hon Lau, S. Rao, N. Taft, and J. D. Tygar. ANTIDOTE: Understanding and defending against poisoning of anomaly detectors. In *Proc. 9th Internet Measurement Conference*, IMC, 2009.

[47] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *Proc. IEEE Security and Privacy Symposium*, S&P, 2017.

[48] N. Srndic and P. Laskov. Mimicus - Contagio Dataset. https://github.com/srndic/mimicus, 2009. Online; accessed 8 May 2017.

[49] N. Srndic and P. Laskov. Practical evasion of a learning-based classifier: A case study. In *Proc. IEEE Security and Privacy Symposium*, S&P, 2014.

[50] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. arXiv:1312.6199, 2014.

[51] D. E. Tyler. Robust statistics: Theory and methods. *Journal of the American Statistical Association*, 103(482):888–889, 2008.

[52] S. Venkataraman, A. Blum, and D. Song. Limits of learning-based signature generation with adversaries. In *Network and Distributed System Security Symposium*, NDSS. Internet Society, 2008.

[53] G. Wang, T. Wang, H. Zheng, and B. Y. Zhao. Man vs. machine: Practical adversarial detection of malicious crowdsourcing workers. In *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA, 2014. USENIX Association.

[54] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli. Is feature selection secure against training data poisoning? In *Proc. 32nd International Conference on Machine Learning*, volume 37 of *ICML*, pages 1689–1698, 2015.

[55] H. Xu, C. Caramanis, and S. Mannor. Robust regression and Lasso. *IEEE Transactions on Information Theory*, 56(7):3561–3574, 2010.

APPENDIX A
THEORETICAL ANALYSIS OF LINEAR REGRESSION

We prove the equivalence of $\mathcal{W}_{\mathrm{tr}}$ and $\mathcal{W}'_{\mathrm{tr}}$ with the following theorem.

Theorem 3. *Consider OLS regression. Let $\mathcal{D}_{\mathrm{tr}} = \{\mathbf{X}, Y\}$ be the original dataset, $\boldsymbol{\theta}_0 = (\boldsymbol{w}_0, b_0)$ the parameters of the original OLS model, and $\mathcal{D}'_{\mathrm{tr}} = \{\mathbf{X}, Y'\}$ the dataset where Y' consists of predicted values from $\boldsymbol{\theta}_0$ on $\mathbf{X}$. Let $\mathcal{D}_p = \{\mathbf{X}_p, Y_p\}$ be a set of poisoning points. Then*

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}_{\mathrm{tr}} \cup \mathcal{D}_p, \boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}'_{\mathrm{tr}} \cup \mathcal{D}_p, \boldsymbol{\theta})$$

Furthermore, we have $\frac{\partial \mathcal{W}_{\mathrm{tr}}}{\partial \boldsymbol{z}} = \frac{\partial \mathcal{W}'_{\mathrm{tr}}}{\partial \boldsymbol{z}}$, where $\boldsymbol{z} = (\boldsymbol{x}_c, y_c)$. Then the optimization problem for the adversary, and the gradient steps the adversary takes, are the same whether $\mathcal{W}_{\mathrm{tr}}$ or $\mathcal{W}'_{\mathrm{tr}}$ is used.

Proof. We begin by showing that

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}'_{\mathrm{tr}}, \boldsymbol{\theta}).$$

By definition, we have $\boldsymbol{\theta}_0 = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta})$. In Y', $y'_i = f(\boldsymbol{x}_i, \boldsymbol{\theta}_0)$, so $\mathcal{L}(\mathcal{D}'_{\mathrm{tr}}, \boldsymbol{\theta}_0) = 0$. But $\mathcal{L} \geq 0$, so $\boldsymbol{\theta}_0 = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}'_{\mathrm{tr}}, \boldsymbol{\theta})$.

We can use this to show that $\mathbf{X}^T Y = \mathbf{X}^T Y'$. Recall that the closed form expression for OLS regression trained on $\mathbf{X}, Y$ is $\boldsymbol{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T Y$. Because $\boldsymbol{\theta}_0$ is the OLS model for both $\mathcal{D}_{\mathrm{tr}}, \mathcal{D}'_{\mathrm{tr}}$, we have

$$(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T Y) = (\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T Y'),$$

but $(\mathbf{X}^T\mathbf{X})^{-1}$ is invertible, so $\mathbf{X}^T Y = \mathbf{X}^T Y'$. We can use this to show that $\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}_{\mathrm{tr}} \cup \mathcal{D}_p, \boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}'_{\mathrm{tr}} \cup \mathcal{D}_p, \boldsymbol{\theta})$ for any $\mathcal{D}_p$. Consider the closed form expression for the model learned on $\mathcal{D}_{\mathrm{tr}} \cup \mathcal{D}_p$:

$$(\mathbf{X}^T\mathbf{X} + \mathbf{X}_p^T X_p)^{-1}(\mathbf{X}^T Y + \mathbf{X}_p^T Y_p) = (\mathbf{X}^T\mathbf{X} + \mathbf{X}_p^T\mathbf{X}_p)^{-1}$$
$$(\mathbf{X}^T Y' + \mathbf{X}_p^T Y_p)$$

which is exactly the model learned on $\mathcal{D}'_{\mathrm{tr}} \cup \mathcal{D}_p$. So the learned models for the two poisoned datasets are the same. Note that this also holds for ridge regression, where the Hessian has a λI term added, so it is also invertible.

We proceed to use $\mathbf{X}^T Y = \mathbf{X}^T Y'$ again to show that $\frac{\partial \mathcal{W}_{\mathrm{tr}}}{\partial \boldsymbol{z}} = \frac{\partial \mathcal{W}'_{\mathrm{tr}}}{\partial \boldsymbol{z}}$.

$$\frac{\partial \mathcal{W}_{\mathrm{tr}}}{\partial \boldsymbol{z}} = \frac{2}{n}(\mathbf{X}\boldsymbol{\theta} - Y)^T \mathbf{X}\frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{z}}$$

$$\frac{\partial \mathcal{W}'_{\mathrm{tr}}}{\partial \boldsymbol{z}} = \frac{2}{n}(\mathbf{X}\boldsymbol{\theta} - Y')^T \mathbf{X}\frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{z}}$$

So the difference between the gradients is

$$\frac{\partial \mathcal{W}'_{\mathrm{tr}}}{\partial \boldsymbol{z}} - \frac{\partial \mathcal{W}_{\mathrm{tr}}}{\partial \boldsymbol{z}} = \frac{2}{n}(Y - Y')^T \mathbf{X}\frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{z}} = \mathbf{0}.$$

Then both the learned parameters and the gradients of the objectives are the same regardless of the poisoned data added. □

We can now perform the derivation of the exact form of the gradient of $\mathcal{W}'_{\mathrm{tr}}$. We have:

$$\frac{\partial \mathcal{W}'_{\mathrm{tr}}}{\partial \boldsymbol{z}} = \frac{2}{n}\sum_{i=1}^{n}((\boldsymbol{w} - \boldsymbol{w}_0)^T \boldsymbol{x}_i + (b - b_0))\left(\boldsymbol{x}_i^T \frac{\partial \boldsymbol{w}}{\partial \boldsymbol{z}} + \frac{\partial b}{\partial \boldsymbol{z}}\right).$$

The right hand side can be rearranged to

$$(\boldsymbol{w} - \boldsymbol{w}_0)^T \left(\boldsymbol{\Sigma}\frac{\partial \boldsymbol{w}}{\partial \boldsymbol{z}} + \mu\frac{\partial b}{\partial \boldsymbol{z}}\right) + (b - b_0)\left(\mu^T \frac{\partial \boldsymbol{w}}{\partial \boldsymbol{z}} + \frac{\partial b}{\partial \boldsymbol{z}}\right),$$

but the terms with gradients can be evaluated using the matrix equations derived from the KKT conditions from Equation 14, which allows us to derive the following:

$$\frac{\partial \mathcal{W}'_{\mathrm{tr}}}{\partial \boldsymbol{x}_c} = \frac{2}{n}((\boldsymbol{w}_0 - \boldsymbol{w})^T M + (b_0 - b)\boldsymbol{w}^T)$$
$$= \frac{2}{n}(f(\boldsymbol{x}_c, \boldsymbol{\theta}) - f(\boldsymbol{x}_c, \boldsymbol{\theta}_0))(\boldsymbol{w}_0 - 2\boldsymbol{w})^T$$
$$\frac{\partial \mathcal{W}'_{\mathrm{tr}}}{\partial y_c} = \frac{2}{n}(f(\boldsymbol{x}_c, \boldsymbol{\theta}) - f(\boldsymbol{x}_c, \boldsymbol{\theta}_0)).$$

APPENDIX B
ANALYSIS OF TRIM ALGORITHM

We present here an analysis on the convergence and estimation properties of the TRIM algorithm.

Convergence. First, Algorithm 2 can be proven to always terminate by the following theorem.

Theorem 1. *Algorithm 2 terminates in a finite number of iterations.*

Proof. We first prove that for each iteration i that does not terminate the loop, we have $R^{(i)} < R^{(i-1)}$. Since each subset of $\{1, ..., n\}$ with size $n-p$ uniquely corresponds to one value R, there is only a finite number of possible R during training. If the algorithm does not terminate, then there will be an infinite long sequence of $R^{(i)}$, contradicting that the set of all possible R is finite.

We only need to show $R^{(i)} \leq R^{(i-1)}$, as the algorithm terminates when $R^{(i)} = R^{(i-1)}$. In fact, we have

$$R^{(i)} = \mathcal{L}(\mathcal{D}^{\mathcal{I}^{(i)}}, \boldsymbol{\theta}^{(i-1)}) \leq \mathcal{L}(\mathcal{D}^{\mathcal{I}^{(i-1)}}, \boldsymbol{\theta}^{(i-1)})$$
$$\leq \mathcal{L}(\mathcal{D}^{\mathcal{I}^{(i-1)}}, \boldsymbol{\theta}^{(i-2)}) = R^{(i-1)}.$$

Parameter	Purpose	Values
η	Line Search Learning Rate	$\{0.01, 0.03, 0.05, 0.1,$
—	—	$0.3, 0.5, 1.0\}$
β	Line Search Learning Rate Decay	$\{0.75\}$
ϵ	Attack Stopping Condition	10^{-5}
λ	Regularization Parameter	Set with Cross Validation

TABLE VI: Description of Parameters for Algorithm 1.

The first inequality is because of the definition of $\mathcal{I}^{(i)}$ (line 8), while the second is due to the definition of $\boldsymbol{\theta}^{(i)}$ (line 9). $\square$

Estimation bound. We now prove Theorem 2. We restate it below.

Theorem 2. *Let $\mathcal{D}_{\mathrm{tr}}$ denote the original training data, $\hat{\boldsymbol{\theta}}$ the global optimum for (17), and $\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta})$ the estimator in the ideal world on pristine data. Assuming $\alpha < 1$, there exist a subset $\mathcal{D}' \subset \mathcal{D}_{\mathrm{tr}}$ of $(1-\alpha)\cdot n$ pristine data samples such that*

$$\mathrm{MSE}(\mathcal{D}', \hat{\boldsymbol{\theta}}) \leq \left(1 + \frac{\alpha}{1-\alpha}\right)\mathcal{L}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta}^\star) \quad (19)$$

Proof. Assume $\hat{\boldsymbol{\theta}} = (\hat{\boldsymbol{w}}, \hat{b}), \hat{\mathcal{I}}$ optimize (17). We have:

$$\mathcal{L}(\mathcal{D}^{\hat{\mathcal{I}}}, \hat{\boldsymbol{\theta}}) \leq \mathcal{L}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta}^\star). \quad (20)$$

Since the adversary can poison at most $\alpha \cdot n$ data points, there exists a subset $\mathcal{I}' \subseteq \hat{\mathcal{I}}$ containing $(1-\alpha)n$ indexes corresponding to pristine data points. We define $\mathcal{D}' = \mathcal{D}^{\mathcal{I}'}$. Thus, we have

$$
\begin{aligned}
\mathcal{L}(\mathcal{D}', \hat{\boldsymbol{\theta}}) &= \frac{1}{(1-\alpha)n} \sum_{(\boldsymbol{x},y)\in\mathcal{D}'} (\hat{\boldsymbol{w}}^T\boldsymbol{x} + \hat{b} - y)^2 + \lambda\Omega(\hat{\boldsymbol{\theta}}) \\
&\leq \frac{1}{(1-\alpha)n} \sum_{(\boldsymbol{x},y)\in\mathcal{D}^{\mathcal{I}}} (\hat{\boldsymbol{w}}^T\boldsymbol{x} + \hat{b} - y)^2 + \lambda\Omega(\hat{\boldsymbol{\theta}}) \\
&= \frac{1}{1-\alpha}\mathcal{L}(\mathcal{D}^{\hat{\mathcal{I}}}, \hat{\boldsymbol{\theta}}) - \frac{1}{1-\alpha}\cdot\lambda\Omega(\hat{\theta}) + \lambda\Omega(\hat{\theta}) \\
&\leq \frac{1}{1-\alpha}\mathcal{L}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta}^\star) - \frac{\alpha}{1-\alpha}\cdot\lambda\Omega(\hat{\boldsymbol{\theta}}) \\
&\leq \left(1 + \frac{\alpha}{1-\alpha}\right)\mathcal{L}(\mathcal{D}_{\mathrm{tr}}, \boldsymbol{\theta}^\star). \quad (21)
\end{aligned}
$$

Notice that in the second step, we apply the fact below:

$$\sum_{(\boldsymbol{x},y)\in\mathcal{D}^{\mathcal{I}}} (\hat{\boldsymbol{w}}^T\boldsymbol{x} + \hat{b} - y)^2 = n[\mathcal{L}(\mathcal{D}^{\mathcal{I}}, \hat{\boldsymbol{\theta}}) - \lambda\Omega(\hat{\boldsymbol{\theta}})]$$

The second to last step is derived by applying Inequality (20), and the last step comes from

$$\lambda\Omega(\hat{\theta}) \geq 0.$$

Further, we have

$$\mathrm{MSE}(\mathcal{D}', \hat{\boldsymbol{\theta}}) \leq \mathcal{L}(\mathcal{D}', \hat{\boldsymbol{\theta}}) \quad (22)$$

By combining (21) and (22), we can get our conclusion. $\square$

In this section, we discuss parameter setting for the baseline attack by Xiao et al. [54]. We perform experiments on the same dataset used by Xiao et al. [54] to test and optimize the baseline attack.

PDF dataset. The PDF malware dataset is a classification dataset containing 5000 benign and 5000 malicious PDF files [48], [49]. It includes 137 features, describing information such as size, author, keyword counts, and various timestamps. We pre-process the data by removing features that were irrelevant (e.g., file name) or had erroneous values (e.g., timestamps with negative values). We also use one-hot encoding for categorical features (replacing the categorical feature with a new binary feature for each distinct value) and apply the logarithm function to columns with large values (e.g., size, total pixels), resulting in 141 features. Each feature is normalized by subtracting the minimum value, and dividing by its range, so that all these features are in [0,1].

Hyperparameters. In order to analyze the baseline attack, we perform an experiment that reproduces exactly the setting from Xiao et al. [54]. We choose a random subset of 300 files for training and a non-overlapping subset of 5000 points for testing the models. To take advantage of our multi-core machines, we parallelize the code by allowing each core to run different instances of the **for** loop body starting on line 6 in Algorithm 1.

There are 3 hyperparameters that control the gradient step and convergence of the iterative search procedure in the algorithm (η, β, and ϵ). The η parameter controls the step size taken in the direction of the gradient. We selected from 7 different values in a wide range, by testing each on 20% poisoning and identifying the value with the largest MSE increase. The β parameter controls the decay of the learning rate, so we take smaller steps as we get closer to the optimal value. We fixed this value to 0.75 and decayed (set $\eta \leftarrow \eta * \beta$) when a step did not make progress. We found this setting to work well on many problems. We fixed the ϵ parameter for attack stopping condition at 0.00001, and choose the λ regularization parameter for the regression model with cross validation. Our parameter settings are detailed in Table VI).
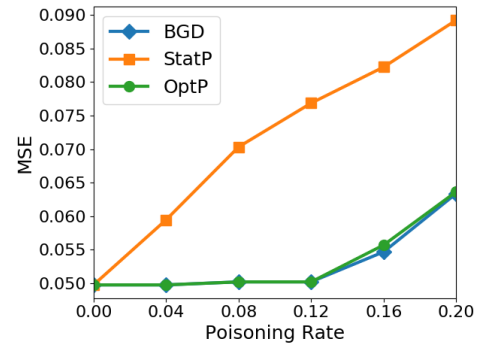

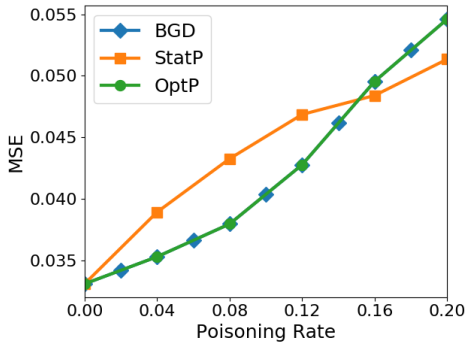Fig. 7: Attack MSE on Contagio dataset for ridge.

Fig. 8: Attack MSE on Contagio dataset for Lasso.

Attack effectiveness. We ran our OptP and StatP attacks on this dataset, in addition to BGD. We expect OptP to be very similar in terms of poisoning to BGD because it is run in the classification setting. Our proposed optimization framework is specific to regression. For instance, in classification settings optimizing by both x and y variables is exactly the same as optimizing only by x. For the initialization strategies, InvFlip and BFlip are exactly the same in the classification setting. In our framework we are exploiting the continuous response variables of regression for designing more effective optimization-based poisoning attacks. The only modification to BGD might come from using $\mathcal{W}_{\mathrm{val}}$ as an optimization objective, but we expect that in isolation that will not produce significant changes. We showed in Section V-B that $\mathcal{W}_{\mathrm{val}}$ is most likely to be effective when optimization by (x, y) is used. Our graphs from Figures 7 and 8 confirm this expectation, and indeed OptP and BGD are very similar in the attack MSEs. The effectiveness of the BGD attack is similar to that reported by Xiao et al. [54] and we have confidence that our implementation and choice of hyper-parameters are accurate. Interestingly, the StatP attack outperforms BGD and OptP by 40% for ridge regression. We believe that pushing the feature values to the boundary as done by StatP has higher effect as a poisoning strategy for ridge regression in which the loss function is convex and the optimization maximum is achieved in the corners. That is not always the case with models such as Lasso, but still StatP is quite effective at poisoning Lasso as well.