Robust Linear Regression Against Training Data Poisoning

Chang Liu
University of California, Berkeley
liuchang@eecs.berkeley.edu

Bo Li
University of California, Berkeley
crystalboli@berkeley.edu

Yevgeniy Vorobeychik
Vanderbilt University
yevgeniy.vorobeychik@vanderbilt.edu

Alina Oprea
Northeastern University
a.oprea@northeastern.edu

ABSTRACT

The effectiveness of supervised learning techniques has made them ubiquitous in research and practice. In high-dimensional settings, supervised learning commonly relies on dimensionality reduction to improve performance and identify the most important factors in predicting outcomes. However, the economic importance of learning has made it a natural target for adversarial manipulation of training data, which we term *poisoning attacks*. Prior approaches to dealing with robust supervised learning rely on strong assumptions about the nature of the feature matrix, such as feature independence and sub-Gaussian noise with low variance. We propose an integrated method for robust regression that relaxes these assumptions, assuming only that the feature matrix can be well approximated by a low-rank matrix. Our techniques integrate improved robust low-rank matrix approximation and robust principle component regression, and yield strong performance guarantees. Moreover, we experimentally show that our methods significantly outperform state-of-the-art robust regression both in running time and prediction error.

1 INTRODUCTION

Machine learning has become widely deployed in a broad array of applications. An important class of machine learning applications enable scalable security defenses, such as spam filtering, traffic analysis, and fraud detection [2, 8, 27]. In these applications, reliability of the machine learning system is crucial to enforce security against powerful adversaries, but strong incentives exist to reduce learning efficacy (e.g., to bypass spam filters).

An important factor in building a reliable machine learning system is the availability of a collection of high-quality training samples. To achieve this, practitioners can either rely on public crowd-sourcing services, such as Amazon Mechanical Turk, or private teams to collect training data sets. However, both of these approaches also open the door to allow adversaries injecting corrupted (poisoned) data points.

On the other hand, recent research demonstrates that existing systems are vulnerable in the presence of adversaries who can manipulate either the training set (i.e., poisoning attacks [28]) or test data (i.e., evasion attacks [19–21]). Consequently, an important agenda in both machine learning and security research is to develop learning algorithms that are robust to data manipulation.

In this work, we focus on designing supervised learning algorithms that are resilient against adversarial poisoning attacks with formal guarantees. Existing research on robust machine learning dates back to algorithms for robust PCA [7]. Most of them assume that a fraction of the underlying dataset is randomly, rather than adversarially, corrupted. Recently, Chen et al. [10] and Feng et al. [14] considered robust regression in face of adversarial corruption. The former considers robust linear regression and the latter logistic regression. However, both make extremely strong assumptions on feature independence and sub-Gaussian distribution per feature with vanishing variance (as $O(\frac{1}{n})$), rendering them impractical and severely limiting the scope of associated theoretical guarantees.

We study a common framework for high-dimensional regression, which proceeds through the following two steps: First, dimension reduction, such as PCA, is performed to project the high-dimensional features into a low-dimensional subspace corresponding to the space where pristine data can be sampled. Second, linear regression is performed to learn the model to best characterize the data.

We consider adversaries who might try to poison and mislead either or both of the two steps, and thus we have two design goals in mind. First, we must make sure that the dimensionality reduction step can reliably recover the low-rank subspace; second, the resulting regression performed on the subspace can recover sufficiently accurate predictions. We aim to achieve these goals despite noise in the dataset and adversarially-poisoned samples. While these problems have previously been considered in isolation, ours is the first integrated approach. More significantly, the effectiveness of our approach relies on far weaker assumptions than prior art (for example, we do not need the typical assumption of sub-Gaussian noise with vanishing variance; see, e.g. [10]), and, as a result, our proposed practical algorithms significantly outperform state-of-the-art alternatives.

Specifically, we assume that labels $\mathbf{y}$ are a linear function of the true feature matrix $\mathbf{X}_\star$ with additive zero-mean noise. In addition,

AISec '17, , November 3, 2017, Dallas, TX, USA
© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN ISBN 978-1-4503-5202-4/17/11…$15.00
https://doi.org/10.1145/3128572.3140447

$\mathbf{X}_\star$ is corrupted with noise, and the adversary subsequently adds a collection of corrupted rows to the training data. In this model, our approach involves two parts: first, we develop a novel robust matrix factorization algorithm which correctly recovers the subspace whenever this is possible, and second, a trimmed principle component regression, which uses the recovered basis and trimmed optimization to estimate linear model parameters.

Our main contributions are as follows:

- **Novel algorithm for robust matrix factorization:** We develop a novel algorithm that reliably recovers the low-rank subspace of the feature matrix despite both noise (about which we make few assumptions) and adversarial examples. We prove that our algorithm is effective if and only if subspace recovery is possible.
- **Novel robust regression algorithm with significantly weaker assumptions:** In contrast to prior robust regression work, we do not require either feature independence or low-variance sub-Gaussian distribution of features. We prove that our algorithm can reliably learn the low-dimensional linear model despite data corruption and noise.
- **Significant improvement in running time and accuracy:** We conduct empirical evaluation and demonstrate that our algorithms significantly outperform prior art in both running time and prediction efficacy.

2 OVERVIEW

Given a dataset $(X_i; y_i)_{i \in [1,n]}$, the machine learning problem is to recover the function f that can best characterize the hidden relationship between X_i and y_i, i.e., $f(X_i)$ and y_i are close. Depending on the values that y_i can take, a machine learning problem is either a classification problem, in which y_i can only take a value from a finite discrete set of "classes", or a regression problem, in which y_i can take values from a continuous space, on which a distance function is well-defined.

In this work, we focus on the regression problem. In particular we consider the most fundamental and widely used machine learning models — linear models. Given a dataset of samples $(\mathbf{X}; \mathbf{y})$, where $\mathbf{y}$ is the dependent variable depending on $\mathbf{X}$, a linear model assumes that $\mathbf{y}$ is "close" to $\beta \cdot \mathbf{X}$ with respect to the distance function, for a set of parameters w.

In this section, we first present the threat model; then we formalize the problem considered in this work; last, we give a road-map of our approach while explaining the paper organization.

2.1 Threat Model

In this work, we assume that the defender collects a set of training data, which can be manipulated by the adversary. The defender thus does not have access to the pristine data before adversarial manipulation. The defender can choose its defense strategy and the training approach. But the defender does not know the attack strategies employed by the adversary.

The attacker has access to the pristine training data, and can insert new data samples whose volume is a fraction of the pristine one up to an upper bound. We assume that the attacker knows the training algorithm used by the defender, and all hyper-parameters that will be used if any.

The above threat model may be stronger than real-world attackers, since in the real world scenarios, the attacker may not be able to know the defense strategy employed, or the entire pristine training data. By assuming such a strong attacker, we can analyze the worst-case performance of the defense algorithm, and thus all security guarantees under our model automatically apply to any weaker threat models based on the Kerckhoffs's principle [26].

This threat model can also be simulated by a zero-sum Stackelberg game, in which the defender chooses the defense strategy first, and then the attacker chooses the attack strategy based on the defender's choice. The defender's goal is to maximize her utility, which is the expected serving time accuracy, while the attacker tries to minimize this utility given a certain budget on the number of training instances that can be poisoned.

2.2 Problem Setup

We start with the pristine training dataset of n labeled examples, $\langle \mathbf{X}_\star; \mathbf{y}_\star \rangle$, which subsequently suffers from two types of corruption: noise is added to feature vectors, and the adversary adds n_1 malicious examples (feature vectors and labels) to best mislead the learning. We assume that the adversary has full knowledge of the learning algorithm. The learner's goal is to learn a model on the corrupted dataset which is similar to the true model. The feature space is high-dimensional, and the learner will perform dimensionality reduction prior to learning. In particular, we assume that $\mathbf{X}_\star$ is low-rank with a basis $\mathbf{B}$, and we assume that the true model is the associated low-dimensional linear regression.

Formally, observed training data is generated as follows:

(1) **Ground truth:** $\mathbf{y}_\star = \mathbf{X}_\star \beta^\star = \mathbf{U}\beta^\star_U$, where $\beta^\star$ is the true model, $\beta^\star_U$ is its low-dimensional representation, and $\mathbf{U} = \mathbf{X}_\star B$ is the low-dimensional embedding of $\mathbf{X}_\star$.
(2) **Noise:** $\mathbf{X}_0 = \mathbf{X}_\star + \mathbf{N}$, where $\mathbf{N}$ is a noise matrix with $\|\mathbf{N}\|_\infty \leq \epsilon; \mathbf{y}_0 = \mathbf{y}_\star + e$, where e is i.i.d. zero-mean Gaussian noise with variance σ.
(3) **Corruption:** The attacker adds n_1 adversarially crafted examples $(\mathbf{x}_a; \mathbf{y}_a)$ to get $\langle \mathbf{X}; \mathbf{y} \rangle$, which maximally skews prediction performance of low-dimensional linear regression.

To formally characterize how well the learner performs in this setting, we define (1) a *model function* $f(\mathbf{X}_0; \mathbf{y}_0)$ which is the model learned on $\langle \mathbf{X}_\star; \mathbf{y}_\star \rangle$; (2) a *loss function* l; and (3) a *threshold function* $\delta(z)$ which takes as input $z > 1$, and is increasing in z. Our metric is $(f; l; \delta)$-*tolerance*:

Definition 1 ($(f; l; \delta)$-tolerance). *We say that learner* L *is* $(f; l; \delta)$-tolerant, *if for any attacker, and any* $z > 1$, *we have*

$$l(L\ (X; y); f(X_0; y_0)) \leq \delta(z)$$

with probability at least $1 - c_1 z^{-c_2}$, *for some constant* $c_1; c_2 > 0$.

In our setting, $f(\mathbf{X}_0; \mathbf{y}_0)$ returns $\beta^\star$ and l is expected quadratic loss $E_x\ (x(\beta - \beta^\star))^2$.

n	Number of pristine samples
n_1	Number of poisoning samples
$n + n_1$	Total number of training samples
γ	The corruption ratio n_1/n
β	Estimated parameter vector
$\beta^\star$	True parameter vector
$\beta^\star_U$	Low-dimensional representation of $\beta^\star$
$\langle \mathbf{X}_\star; \mathbf{y}_\star \rangle$	Pristine training data
$\langle \mathbf{X}_0; \mathbf{y}_0 \rangle$	Examples with Gaussian uncertainty
$\langle \mathbf{x}_a; \mathbf{y}_a \rangle$	Adversarially crafted examples
$\langle \mathbf{X}; \mathbf{y} \rangle$	Dataset after injecting poisoning instances
$\mathbf{B}$	Basis for $X_\star$
f	Learning model
l	Loss function

Table 1: Notation Table

For convenience, we let O denote the set of (unknown) indices of the samples in $\mathbf{X}$ coming from $\mathbf{X}_0$ and $A = \{1; :::; n + n_1\} - O$ the set of indices for adversarial samples in $\mathbf{X}$. For an index set I and matrix M, M^I denotes the sub-matrix containing only rows in I; similar notation is used for vectors. We define $\gamma = \frac{n_1}{n}$ as the *corruption ratio*, or the ratio of corrupted and pristine data.

All solutions presented in this work assume that n is given. In practice, we only need estimate n as a lower bound on the number of pristine samples. When there are n' pristine samples where $n' > n$, we can simply consider this case as the adversary has $n' - n$ unused budget during poisoning, and our analysis still holds in such a case.

We summarize the notations used in the paper in Table 1.

2.3 Solution overview

Our goal is to design a learner L to estimate the coefficients $\hat{\beta}$ of the true model $\beta^\star$ using low-dimensional embedding of a high-dimensional model. We achieve this goal in two steps: (1) recover the subspace $\mathbf{B}$ of $\mathbf{X}_\star$; (2) project $\mathbf{X}$ onto $\mathbf{B}$, and estimate $\hat{\beta}$ using robust principle component regression. The key challenge is that an adversary can design corrupted data to interfere with both the first and second steps of the process.

For the first step (Section 3), we develop a *robust subspace recovery* algorithm which can account for both noise $\mathbf{N}$ and adversarial examples in correctly recovering the subspace of $\mathbf{X}_\star$. We characterize necessary and sufficient conditions for successful subspace recovery, showing that our algorithm succeeds whenever recovery is possible. The challenge in the second step (Section 4) is that the adversary can construct $\mathbf{X}^A$ from the same subspace as $\mathbf{X}_\star$, but with the different distribution of $\langle \mathbf{X}^A; \mathbf{y}^A \rangle$ from $\langle \mathbf{X}_\star; \mathbf{y}_\star \rangle$. To address this, we propose the *trimmed principle component regression* algorithm to minimize the loss function over only a subset of the dataset ensuring that the adversary can have only a limited impact by adding n_1 arbitrary corrupted samples without having these instances being discarded. Our theoretical results demonstrate that the combined approach is an $(f; l; \delta)$*-tolerant* learning algorithm. Finally, in Section 5, we present an efficient practical implementation of our methods, which we evaluate in Section 6.

3 ROBUST SUBSPACE RECOVERY

In this section, we discuss how to recover the low-rank subspace of $\mathbf{X}_\star$ from $\mathbf{X}$. Our goal is to exactly recover the low-rank subspace, i.e., returning a basis for $\mathbf{X}_\star$. We show sufficient and necessary conditions for this problem to be solvable, and provide algorithms when this is possible. As a warmup, we first discuss the noise-free version of the problem, and then present our results for the case when noise is added to training data. Proofs of the theorems presented in this section can be found in Appendix A. Formally, we consider the following problem:

Problem Definition 1 (Subspace Recovery). *Design an algorithm $L_{recovery}$, which takes as input X, and returns a set of vectors B which form the basis of $X_\star$.*

3.1 Warmup: Noise-free Subspace Recovery

We first consider an easier version of Problem 1 with $N = 0$. In this case, we know that $\mathbf{X}^O = \mathbf{X}_\star$. We assume that we know $\text{rank}(\mathbf{X}_\star) = k$ (or have an upper bound on it). Below we demonstrate that there exists a sharp threshold θ on n_1 such that whenever $n_1 < \theta$, we can solve Problem 1 exactly with high probability, whereas if $n_1 \geq \theta$, Problem 1 cannot be solved. To characterize this threshold, we define the cardinality of the *maximal rank $k - 1$ subspace MS_{k-1}*$(\mathbf{X}_\star)$ as the optimal value of the following problem:

$$\max_I |I| \text{ s.t. } \text{rank}(\mathbf{X}_\star^I) \leq k - 1$$

Intuitively, the adversary can insert $n_1 = n - MS_{k-1}(\mathbf{X}_\star)$ samples to form a rank k subspace, which does not span $\mathbf{X}_\star$. The following theorem shows that in this case, there is indeed no learner that can successfully recover the subspace of $\mathbf{X}_\star$.

Theorem 1. *If $n_1 + MS_{k-1}(X_\star) \geq n$, then there exists an adversary such that no algorithm $L_{recover}$ solves Problem 1 with probability greater than* $1/2$.

On the other hand, when n_1 is below this threshold, we can use Algorithm 1 to recover the subspace of $\mathbf{X}_\star$.

```
Algorithm 1 Exact recover algorithm for Problem 1 (Noisy-free)
    We search for a subset $I$ of indices, such that $|I| = n$, and
    $\text{rank}(\mathbf{X}^I) = k$
    return a basis of $\mathbf{X}^I$.
```

In fact, we can prove the following theorem.

Theorem 2. *If $n_1 + MS_{k-1}(X_\star) < n$, then Algorithm 1 solves Problem 1 for any adversary.*

Theorems 1 and 2 together give the necessary and sufficient conditions on when Problem 1 is solvable, and Algorithm 1 provides a solution. We further show an implication of these theorems on the corruption ratio γ. We can prove that $MS_{k-1}(\mathbf{X}_\star) \geq k - 1$ (see Appendix A). Combining this with Theorem 1, we obtain the following upper bound on γ.

Corollary 1. *If $\gamma \geq 1 - \frac{k-1}{n}$, then Problem 1 cannot be solved.*

3.2 Dealing with Noise

We now consider Problem 1 with noise. Before we discuss the adversary, we first need to assume that the uncorrupted version is solvable. In particular, we assume that $X_\star$ is the unique optimal solution to the following problem:

$$\min_{\mathbf{X}'} ||\mathbf{X}_0 - \mathbf{X}'|| \tag{1a}$$

$$\text{s.t. } \operatorname{rank}(\mathbf{X}') \leq k. \tag{1b}$$

Unless otherwise mentioned, we use $|| \cdot ||$ to denote the Frobenius norm. We put no additional restrictions on $\mathbf{N}$ except above. Note that this assumption is implied by the classical PCA problem [13, 15, 16]. We focus on the optimal value of the above problem, which we term the *noise residual*, denoted as $NR(\mathbf{X}_0) = \mathbf{N}$. Noise residual is a key component to characterize the necessary and sufficient conditions for the solvability of Problem 1.

Characterization of the defender's ability to accurately recover the true basis $\mathbf{B}$ of $\mathbf{X}_\star$ after the attacker adds n_1 malicious instances stems from the attacker's ability to mislead the defender into thinking that some other basis, $\bar{\mathbf{B}}$, better represents $\mathbf{X}_\star$. Intuitively, since the defender does not know $\mathbf{X}_0$, $\mathbf{X}_\star$, or which n_1 rows of the data matrix $\mathbf{X}$ are adversarial, this comes down to the ability to identify the $n - n_1$ rows that correspond to the correct basis (note that it will suffice to obtain the correct basis even if some adversarial rows are used, since the adversary may be forced to align malicious examples with the correct basis to evade explicit detection). As we show below, whether the defender can succeed is determined by the relationship between the noise residual $NR(\mathbf{X}_0)$ and *sub-matrix residual*, denoted as $SR(\mathbf{X}_0)$, which we define as the value optimizing the following problem:

$$\min_{\mathcal{I}, \bar{\mathbf{B}}, \mathbf{U}} ||\mathbf{X}_0^{\mathcal{I}} - \mathbf{U}\bar{\mathbf{B}}|| \tag{2a}$$

$$\text{s.t.} \quad \operatorname{rank}(\bar{\mathbf{B}}) = k, \bar{\mathbf{B}}\bar{\mathbf{B}}^T = I_k, \mathbf{X}_\star \bar{\mathbf{B}}^T \bar{\mathbf{B}} \neq \mathbf{X}_\star \tag{2b}$$

$$\mathcal{I} \subseteq \{1, 2, \ldots, n\}, |\mathcal{I}| = n - n_1. \tag{2c}$$

We now explain the above optimization problem. U and $\bar{\mathbf{B}}$ are $(n - n_1) \times k$ and $k \times m$ matrixes separately. Here $\bar{\mathbf{B}}$ is a basis which the attacker "targets"; for convenience, we require $\bar{\mathbf{B}}$ to be orthogonal (i.e., $\bar{\mathbf{B}}\bar{\mathbf{B}}^T = I_k$). Since the attacker succeeds only if they can induce a basis different from the true $\mathbf{B}$, we require that $\bar{\mathbf{B}}$ does not span of $\mathbf{X}_\star$, which is equivalent to saying $\mathbf{X}_\star \bar{\mathbf{B}}^T \bar{\mathbf{B}} \neq \mathbf{X}_\star$. Thus, this optimization problem seeks $n - n_1$ rows of $\mathbf{X}_\star$, where $\mathcal{I}$ is the corresponding index set. The objective is to minimize the distance between $\mathbf{X}_0^{\mathcal{I}}$ and the span space of the target basis $\bar{\mathbf{B}}$, (i.e., $||\mathbf{X}_0^{\mathcal{I}} - \mathbf{U}\bar{\mathbf{B}}||$).

Algorithm 2 Exact recovery algorithm for Problem 1

```
Solve the following optimization problem and get $\mathcal{I}$.
    $\min_{\mathcal{I}, L} ||\mathbf{X}^{\mathcal{I}} - L||$
    s.t. $\operatorname{rank}(L) \leq k, \mathcal{I} \subseteq \{1, \ldots, n + n_1\}, |\mathcal{I}| = n$        (3)
return a basis of $\mathbf{X}^{\mathcal{I}}$.
```

To understand the importance of $SR(\mathbf{X}_0)$, consider Algorithm 2 for recovering the basis of $\mathbf{X}_\star$, $\mathbf{B}$. If the optimal objective value of optimization problem (2), $SR(\mathbf{X}_0)$, exceeds the noise $NR(\mathbf{X}_0)$, it follows that the defender can obtain the correct basis $\mathbf{B}$ using Algorithm 2, as it yields a better low-rank approximation of $\mathbf{X}$ than any other basis. Else, it is, indeed, possible for the adversary to induce an incorrect choice of basis. The following theorem formalizes this argument.

Theorem 3. *If $SR(X_0) \leq NR(X_0)$, then no algorithm can recover the exact subspace of $X_\star$ with probability greater than* $1/2$. *If $SR(X_0) > NR(X_0)$, then Algorithm 2 solves Problem 1.*

To draw connection between the noisy case and the noise-free case, we can view Theorem 1 and 2 as special cases of Theorem 3.

Theorem 4. *When $N = 0$, $SR(X_0) > NR(X_0) = 0$ if and only if $n_1 + MS_{k-1}(X_\star) < n$.*

4 TRIMMED PRINCIPAL COMPONENT REGRESSION

In this section, we present trimmed principal component regression (T-PCR) algorithm. The key idea is to leverage the principal component regression (PCR) approach to estimate β, but during the process trimming out those malicious samples that try to deviate the estimator from the true ones. In the following, we present the approach, which is similar to the standard PCR approach, though we do not require computing the exact singular vectors of $\mathbf{X}_\star$.

Assume we recover a basis $\mathbf{B}$ of $\mathbf{X}_\star$. Without loss of generality, we assume that $\mathbf{B}$ is an orthogonal basis of k row vectors. Since $\mathbf{B}$ is a basis for $\mathbf{X}_\star$, we assume $\mathbf{X}_\star = \mathbf{U}_\star \mathbf{B}$. Then we know that, by optimization (1), $\mathbf{U}_\star = \operatorname{argmin}_U ||\mathbf{X}_0 - U\mathbf{B}||$. We compute $\mathbf{U} = \operatorname{argmin}_U ||\mathbf{X} - U\mathbf{B}||$, and, by definition, we know $\mathbf{U}_\star = \mathbf{U}^{\mathcal{O}}$. By OLS estimator, we know that $\mathbf{U}^T = (\mathbf{B}\mathbf{B}^T)^{-1}\mathbf{B}\mathbf{X}^T$, and thus $\mathbf{U} = \mathbf{X}\mathbf{B}^T$.

To estimate $\mathbf{y} = \mathbf{X}_\star \beta + e$, we assume $\beta_U = \mathbf{B}\beta$. Since $\mathbf{X}_\star = \mathbf{U}_\star \mathbf{B}$, we convert the estimation problem of β from a high dimensional space to the estimation problem of β_U from a low dimensional space, such that $y = \mathbf{U}\beta_U + e$. After estimating for β_U, we can convert it back to get $\beta = \mathbf{B}\beta_U$. Notice that this is similar to principal component regression [17].

However, the adversary may corrupt n_1 rows in $\mathbf{U}$ to fool the learner to make a wrong estimation on β_U, and thus on β. To mitigate this problem, we design Algorithm 3. Intuitively, during

Algorithm 3 Trimmed Principal Component Regression

```
Input: $\mathbf{X}, \mathbf{y}$
(1) Use Algorithm 2 to compute a basis from $\mathbf{X}$, and orthogonalize it to get $\mathbf{B}$
(2) Project $\mathbf{X}$ onto the span space of $\mathbf{B}$ and get $\mathbf{U} \leftarrow \mathbf{X}\mathbf{B}^T$
(3) Solve the following minimization problem to get $\beta_U$
        $\min_{\beta_U} \sum_{j=1} \{(y_i - u_i \beta_U)^2 \text{ for } i = 1, \ldots, n + n_1\}_{(j)}$        (4)
    where $z_{(j)}$ denotes the $j$-th smallest element in sequence $z$.
(4) return $\beta \leftarrow \mathbf{B}\beta_U$.
```

the training process, we trim out the top n_1 samples that maximize the difference between the observed response y_i and the predicted response $u_i\beta_U$, where u_i denotes the i-th row of U. Since we know the variances of these differences are small (i.e., recall Section 2, σ is the variance of the random noise $y - x\beta^\star$), these samples corresponding to the largest differences are more likely to be the adversarial ones.

Next, we theoretically bound the prediction differences between our model and the linear regression model learnt on $\mathbf{X}_\star, \mathbf{y}_\star$.

Lemma 1 (TPCR Lemma). *Algorithm 3 returns $\hat{\beta}$, such that for any real value $h > 1$ with at least $1 - ch^{-2}$ probability for some constant c, we have*

$$E_x\left[(x(\hat{\beta} - \beta^\star))^2\right] \le 4\sigma^2\left(1 + \sqrt{\frac{1}{1-\gamma}}\right)^2 \log c \qquad (5)$$

We explain the intuition of this Lemma, and defer the detailed proof to Appendix B. If an adversary wants to fool Algorithm 3, it needs to generate samples (u_i, y_i), such that the loss function $(y_i - u_i\hat{\beta}_U)^2$ is among the smallest n. Since for samples from $\mathbf{X}_\star, \mathbf{y}_\star$, these loss functions are already bounded according to σ, the adversary does not have an ability to significantly skew the estimator. In particular, if $\sigma = 0$, i.e., there is no error while generating $\mathbf{y}_0$ from $\mathbf{X}_\star$, then the adversary can do nothing when $\gamma < 1$, and thus the estimator is the same as the linear regression's estimator on the uncorrupted data.

As an immediate consequence of Lemma 1, we have

Theorem 5. *Given $\delta(c) = 4\sigma^2\left(1 + \sqrt{\frac{1}{1-\gamma}}\right)^2 \log c$, Algorithm 3 is $(f, l, \delta(c))$-tolerant.*

5 PRACTICAL ALGORITHMS

Algorithms 1, 2, and 3 require enumerating a subset of indices, and are thus all exponential time. To make our approach practical, we develop efficient implementations of Algorithms 2 and 3.

5.1 Efficient Robust Subspace Recovery

Consider the objective function (3). Since $\text{rank}(L) \le k$, we can rewrite $L = \mathbf{U}\mathbf{B}^T$ where $\mathbf{U}$'s and $\mathbf{B}$'s shapes are $n \times k$, and $m \times k$ respectively. Therefore, we can rewrite objective (3) as

$$\min_{I,\mathbf{U},\mathbf{B}} ||\mathbf{X}^I - \mathbf{U}\mathbf{B}^T|| \text{ s.t. } |I| = n$$

which is equivalent to

$$\min_{\mathbf{U},\mathbf{B}} \sum_{j=1}^{n} \{||x_i - u_i\mathbf{B}^T|| \text{ for } i = 1, \ldots, n + n_1\}_{(j)} \qquad (6)$$

where x_i and u_i denote the ith row of $\mathbf{X}$ and $\mathbf{U}$ respectively. Solving Objective 6 can be done using alternating minimization, which iteratively optimizes the objective for $\mathbf{U}$ and $\mathbf{B}$ while fixing the other. Specifically, in the wth iteration, we optimize for the following two objectives:

$$\mathbf{U}^{w+1} = \text{argmin}_U ||\mathbf{X} - U(\mathbf{B}^w)^T||$$

Algorithm 4 Trimmed Optimization

```
(1) Randomly assign $\tau_i \in \{0, 1\}$ for $i = 1, \ldots, n + n_1$, such that $\sum_{i=1}^{n+n_1} \tau_i = n$
(2) Optimize $\theta \leftarrow \text{argmin}_\theta \sum_{i=1}^{n+n_1} \tau_i l(y_i, f_\theta(x_i))$;
(3) Compute $\text{rank}_i$ as the rank of $l(y_i, f_\theta(x))$ in the ascending order;
(4) Set $\tau_i \leftarrow 1$ for $\text{rank}_i \le n$, and $\tau_i \leftarrow 0$ otherwise;
(5) Go to 2 if any of $\tau_i$ changes;
(6) return $\theta$.
```

$$\mathbf{B}^{w+1} = \text{argmin}_B \sum_{j=1}^{n} \{||x_i - u_i^{w+1}B^T|| \text{ for } i = 1, \ldots, n + n_1\}_{(j)}.$$

Notice that the second step computes the entire $\mathbf{U}$ regardless of the sub-matrix restriction. This is because we need the entire $\mathbf{U}$ to be computed to update $\mathbf{B}$. The key challenge is to compute $\mathbf{B}^{w+1}$ in each iteration, which is, again, a trimmed optimization problem. The next section presents a scalable solution for such problems.

5.2 Efficient Algorithm for Trimmed Optimization Problems

Both robust subspace recovery and optimizing for (4) rely on solving optimization problems in the form

$$\min_\theta \sum_{j=1}^{n} \{l(y_i, f_\theta(x_i)) \text{ for } i = 1, \ldots, n + n_1\}_{(j)}$$

where $f_\theta(x_i)$ computes the prediction over x_i using parameter θ, and $l(\cdot, \cdot)$ is the loss function. We refer to such problems as *trimmed optimization problems*. It is easy to see that solving this problem is equivalent to solving

$$\begin{array}{c}\min_{\theta, \tau_1, \ldots, \tau_{n+n_1}} \sum_{i=1}^{n+n_1} \tau_i l(y_i, f_\theta(x_i)) \\ \text{s.t. } 0 \le \tau_i \le 1, \sum_{i=1}^{n+n_1} \tau_i = n\end{array}$$

We can use alternating minimization technique to solve this problem, by optimizing for θ, and τ_i respectively. We present this in Algorithm 4. In particular, the algorithm iteratively seeks optimal arguments for θ and $\tau_1, \ldots, \tau_{n+n_1}$ respectively. Optimizing for θ is a standard least square optimization problem. When optimizing $\tau_1, \ldots, \tau_{n+n_1}$, it is easy to see that $\tau_i = 1$ if $l(y_i, f_\theta(x_i))$ is among the largest n; and $\tau_i = 0$ otherwise. Therefore, optimizing for $\tau_1, \ldots, \tau_{n+n_1}$ is a simple sorting step. While this algorithm is not guaranteed to converge to a global optimal, in our evaluation,we observe that a random start of τ typically yields near-optimal solutions.

6 EVALUATIONS

In this section, we evaluate our approach, i.e., T-PCR. First, we evaluate the two components, subspace recovery algorithms and the regression algorithms separately. We employ synthetic datasets to evaluate both the runtime and the effectiveness of the T-PCR approach with the previous state-of-the-art. Second, we evaluate the effectiveness of the entire algorithm using a real-world dataset.

In the following, we first present the setup of our evaluation, and then present the results for each experiment.

6.1 Setup

We implement all algorithms. All programs are run on a workstation with a Intel i7-6800K CPU running at 3.4GHz, 128G memory, and 1TB SSD hard drive. In the following, we explain the baselines used for comparison, datasets, and poisoning strategies.

6.1.1 Implementation details. We implement our defense strategies based on Algorithm 4 discussed in Section 5.

6.1.2 Baseline. We compare our approach with the state-of-the-art alternatives in the literature. For the subspace recovery problem, we compare to two approaches: Chen et al. [11] and Xu et al. [30].

For the end-to-end linear regression problem, we compare our T-PCR algorithm with the recent robust regression approach [10] and standard ridge regression algorithm.

6.1.3 Datasets. In this work, we use two classes of datasets to evaluate our approach: synthetically generated data, and real data. We explain them below.

Synthetic Datasets. We generate datasets using a routine with hyper-parameters n, n_1, k, and m, which represent the number of samples in total, poisoned data samples, intrinsic rank of the data, and the number of features. We set $m = 400$ for all experiments, but n, n_1, and k can be varied to evaluate different aspects of our approach.

For a given $(k; n)$, we generate $\mathbf{X}_\star$ as follows: sample two matrices $\mathbf{U}; \mathbf{B}$ with shape $n \times k$ and $k \times m$ respectively. Each element is sampled independently from a Gaussian distribution $\mathcal{N}(0; 1)$. Once a matrix (e.g., $\mathbf{U}$ or $\mathbf{B}$) is sampled, we verify that the matrix has rank k; or otherwise, we will keep re-sampling the matrix until the matrix has rank k. Once both $\mathbf{U}$ and $\mathbf{B}$ are sampled, and we set $\mathbf{X}_\star = \mathbf{UB}$. We do not add noise to $\mathbf{X}_\star$, unless explicitly stated. When noise is added, the pristine data is generated as $\mathbf{X}_\star + \mathbf{N}$, where each element in $\mathbf{N}$ is randomly sampled from the Gaussian distribution $\mathcal{N}(0; 0{:}01)$.

Real-world Dataset: malicious domains. We obtained a dataset including HTTP logs collected from a large enterprise spanning a period of four months (February, March, July, and August) in 2015. The dataset includes features extracted from the inbound and outbound HTTP and HTTPs communications captured at the border of the enterprise by web proxies. Each log event includes fields from the HTTP headers of the connections, e.g., connection timestamp, source and destination IP address, contacted domain and URL, result code, HTTP action, web referrer, user-agent string and bytes sent and received.

Some filters were applied to the raw datasets to eliminate popular web sites (domains contacted by more than 50 hosts), domains involved in advertisement or CDNs, and also domains with only limited number of connections (less than 5). For the remaining domains, the dataset includes 91 features typically used in security applications for flagging malicious communications. The features extracted from the proxy logs belong to several categories: *Communication structure* (e.g., number of hosts contacting the domain, total number of connections, bytes sent and received to the domain, number of POST, GET connections); *Domain structure* (e.g., number of levels in the domain, number of sub-domains on the same second-level domain and domain name length); *URL structure* (e.g., number of distinct URLs, URL path length and depth, number of parameters, number of values per parameter); *User-agent string features* (e.g., total number of UAs, popularity of the UA across the enterprise, ratio of UA across hosts); *Web referrer features* (e.g., total number of referer domains, fraction of connections without referer); *Result-code features* (e.g., number of successful and failed connections); *Content-type features* (e.g., number of distinct content types).

At the same time, a number of features extracted from publicly available external sources are added. They include: *WHOIS-related features* (e.g., domain age defined as the time since the domain was registered, registration validity defined as time until registration expires), *geographical location* (number of countries and ASNs of the IP address of the domain. For labeling the domains in the datasets, a cloud-based anti-virus engine (VirusTotal) was used. We consider *malicious* all domains flagged by at least three anti-virus engines in VirusTotal. We consider *benign* the domains with a score of 0 on VirusTotal that are in top 100K according to Alexa ranking. Domains with score 1 or 2 are considered *unknown* and removed from the dataset.

For the evaluation purpose, we use the data from Feburary, March, and July as the training set, and the data from August as the test set.

6.1.4 Poisoning Strategy. We employ two poisoning strategies for attacking the subspace recovery problem and the linear regression problem respectively. We present them below.

Poisoning strategy for subspace recovery. We evaluate different approaches for the subspace recovery problem using only synthetic data. Thus, we assume the adversary has access to $\mathbf{X}_\star$, the generated pristine data. We generate corruptions $\mathbf{X}^{\mathcal{A}}$ also as a low rank matrix by generating $\mathbf{U}^{\mathcal{A}}$ and $\mathbf{B}^{\mathcal{A}}$ in the same way as generating $\mathbf{U}$ and $\mathbf{B}$, where $\mathbf{U}^{\mathcal{A}}$ has n_1 rows.

For $\mathbf{B}^{\mathcal{A}}$, we set the first half of $\mathbf{B}^{\mathcal{A}}$ by choosing $k/2$ rows of $\mathbf{X}_\star$, and generating the remaining $k/2$ rows randomly, while ensuring that $\mathbf{B}$ has rank k. We concatenate $\mathbf{X}_\star$ and $\mathbf{X}^{\mathcal{A}} = \mathbf{U}^{\mathcal{A}}\mathbf{B}^{\mathcal{A}}$ to get the matrix with $n + n_1$ rows, and then shuffle them. In doing so, we know that $\mathbf{X}^{\mathcal{A}}$ shares a common subspace of rank $k/2$ with $\mathbf{X}_\star$, but the two subspaces are still different.

Notice that this strategy is designed to trigger the worst case performance for our algorithms. Later, in the evaluation, we will demonstrate that although this strategy does not leverage the information of the baseline algorithms, our approach outperform the baseline approaches.

Poisoning strategy for linear regression. We employ Xiao et al.. [28], the state-of-the-art poisoning strategy for linear models

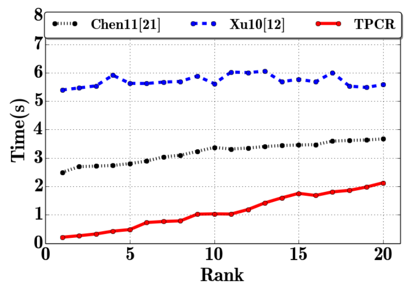


Figure 1: Runtime comparison on the subspace recovery problem

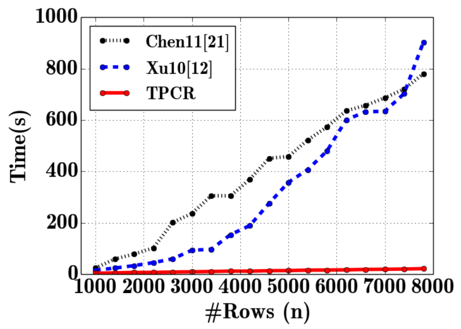


Figure 2: Runtime comparison on the linear regression problem

and variants to create the adversarial labels. The basic idea is to move the data samples along the direction to maximally modify the learned estimator. This process is repeated multiple iterations until the learned model cannot predict correctly. We refer the readers to [28] for more details.

6.2 Subspace recovery

6.2.1 Runtime. We evaluate the runtime of our approach in comparison with baselines by varying the intrinsic rank k and the number of pristine data n.

First, we set $n = 350$; $n_1 = 50$ and vary k from 1 to 20 with the step to be 1. For each k, we evaluate the runtime of our algorithm along with the two baselines [11] and [30]. The results are plotted in Figure 1. Our algorithm is significantly faster than [11] and [30]. The reason is our algorithm is designed in the alternative minimization fashion, so that its runtime is linear to the size of the matrix, and also linear to the intrinsic rank k. On the other hand, both [11] and [30] employ SVD as its sub-routine, which is very time-consuming. The SVD algorithm thus dominants the runtime of both [11] and [30]. Notice that the SVD algorithm's runtime does not rely on k. This is also the reason why these two baseline approaches' runtime does not increase as fast as our approache's when k is increased.

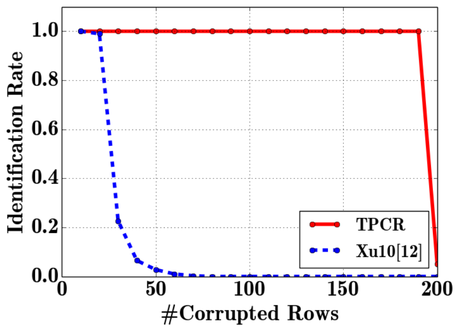


Figure 3: Effectiveness on the subspace recover task using the rate of correct identification of corrupted rows.

Second, we fix $n_1 = 50$ and $k = 20$, and vary n from 1000 to 8000 with the step to be 400. The results of this experiment are presented in Figure 2. We can observe that as long as the pristine training samples increase, our algorithm's runtime increases slowly, while the two baseline approaches are slowing down dramatically. As we have explained above, this is due to that the SVD algorithm dominants most of the runtime of [11] and [30], and the SVD algorithm's runtime increases dramatically when the total number of rows is increased.

From these two comparisons, we conclude that our approach is much more efficient than the baseline approaches and scales well in terms of both the intrinsic rank and the training data volume.

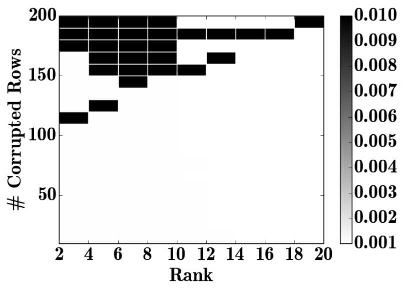
6.2.2 Effectiveness of the subspace recovery algorithm. We study the effectiveness of our subspace recovery algorithm using two metrics. First, we evaluate the percentage of the corrupted rows that can be identified. Intuitively, if a high percentage of the corrupted rows can be identified, then the defender can simply remove those rows to recover the original pristine data.

Second, we evaluate the distance between the recovered matrix and the pristine matrix on noisy data. The smaller this distance is, the more effective the recovery algorithm is.

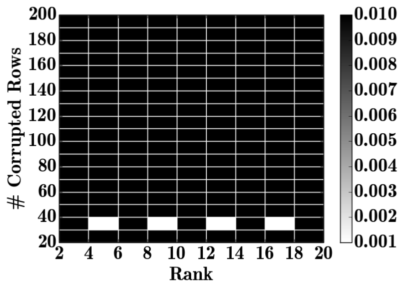
Identification rate of corrupted rows. We compare our algorithm with the two baselines [11, 30] on *the identification rate of corrupted rows*. That is the ratio of the of corrupted rows that can be identified in the total number of rows. We fix $k = 10$, and vary n_1 from 10 to 200 with a step size 10, and $n = 400 - n_1$ to keep $n + n_1 = 400$.

The results are presented in Figure 3. Our approach matches the upper bound of γ provided by Corollary 1. In particular, when $n_1 = n\gamma \leq n(1 - k/n) = n - k$, which means $n_1 \leq 190$, the identification rate of our algorithm is 100%. In this case, our approach can perfectly identify the corrupted rows with 100% accuracy. On the other hand, when $n_1 > 190$, our approach fails, as expected.

In this experiment, the algorithms used in [30] and [11] are identical, and we refer to both as Xu et al. [30]. We can observe that



(a) TPCR



(b) Xu et al. [30]

Figure 4: Effectiveness on the robust regression task

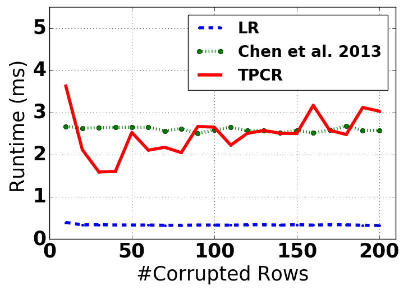


Figure 5: Runtime comparison for robust regression

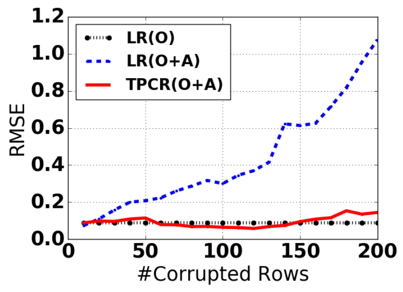


Figure 6: RMSE comparison for robust regression

the identification rate plummets for $n_1 \geq 20$, even though only 5% of the rows are corrupted.

RMSE. We then evaluate our approach's effectiveness on the noisy data. In particular, we add noise to the dataset, and employ the poisoning strategy discussed in Section 6.1.4, which is designed against our approach. Since [11] cannot handle noise, we only compare with [30]. We evaluate the distance between the ground truth matrix and the recovered matrix, i.e., using *the root of mean square (RMSE)* metric of the residual matrix. This metric is used by [30] as well.

Figure 4a and 4b show the RMSE of the difference from recovered $\mathbf{X}^O$ and the true $\mathbf{X}_\star$. We use the grayscale to denote the RMSE: lighter color corresponds to smaller RMSE. On most test cases our algorithm successfully recovers the true subspace, while [30] fails on most cases. Particularly, when $n_1 < 120$, our approach can completely recover the underlying low-rank matrix. When n_1 increases, the condition $SR(\mathbf{X}_0) > NR(\mathbf{X}_0)$ might not hold true, and Theorem 3 says that no algorithm can recover the true subspace with probability greater than 1/2. However, this theorem does not prevent our algorithm succeeding with probability $< 1/2$, which is why we observe several white spots for high n_1.

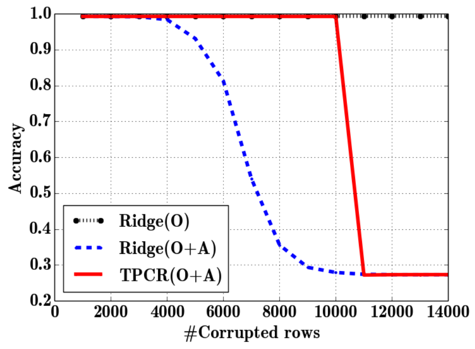
6.3 Robust regression

In this section, we evaluate the trimmed regression component (Algorithm 3) in comparison with baseline approaches, i.e., vanilla linear regression and Chen et al. 2013 [10], which is the only alternative method for linear regression that is designed to be robust to the adversarial data poisoning.

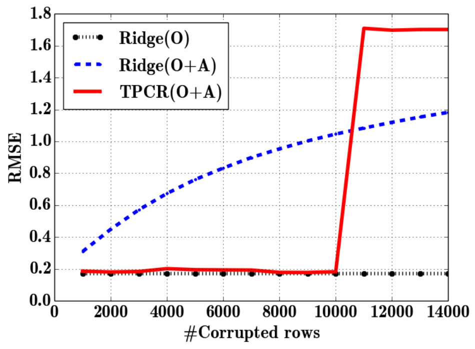
We eliminate the effect of dimension reduction, and set $m = k = 20$ for evaluation purpose. Thus, the generated feature matrix X is guaranteed to be of full rank. We set $n + n_1 = 400$, and vary n_1 from 10 to 200 with a step size 10. In this setting, we choose the Robust Thresholding Regression algorithm proposed by Chen et al. 2013, which is designed for this setting. We evaluate both runtime and the effectiveness.

Runtime comparison. The runtime comparison results are presented in Figure 5. As we can observe, our T-PCR approach's runtime is almost the same as Chen et al. 2013's runtime, though our approach's runtime is not as stable as Chen et al. 2013's. This is because our approach uses an iterative algorithm, and its convergence rate relies on both the initial guess on the poisoning data samples, and the distribution of the poisoning data samples themselves. Thus different poisoned dataset may result in different runtime. In contrast, Chen et al. 2013's approach is deterministic, and its runtime does not depend on the data at all, and thus its performance is more stable.

When compared with vanilla linear regression, both of our approach and Chen et al. 2013 are around 7× slower. This is reasonable, since our approach employs the algorithm solving linear regression in each iteration. Further, we observe that the overall runtime of our



(a) Accuracy



(b) RMSE

Figure 7: Evaluation on a real-world dataset.

approach is at the order of milliseconds. Therefore, we argue that such a slow-down over vanilla linear regression does not hinder our approach's practicality, considering that the matrix factorization component will take a much longer time.

Effectiveness. We evaluate the effectiveness of the robust regression algorithms by evaluating the root-of-mean-square (RMSE) of the prediction using the learned model on test dataset. The results are presented in Figure 6. As a baseline, we also present results for standard OLS linear regression with and without adversarial instances (LR(O+A) and LR(O), respectively). Results are evaluated using a ground truth test set not used for training. The results demonstrate that our algorithm significantly outperforms the alternatives. Surprisingly, Chen et al. 2013's approach results in an orders of magnitude larger RMSE, thus we omit these results from the figure. We can observe that our method works nearly as well as linear regression *without* adversarial instances!

6.4 End-to-end evaluation on Real World Dataset

We evaluate our end-to-end approach combining both subspace recovery algorithms and trimmed regression algorithms over the real world malicious domain dataset. As explained in Section 6.1.3, we use three months (Feburary, March, and July) of data for training, and one month (August) of data for testing. We poison the data using Xiao et al. [28]. This data set contains 18,000 samples in total.

We choose Ridge regression as our baseline, since in our evaluation on synthetic data Chen et al. 2013 did not prove effective. Further, we include the results using Ridge regression trained on pristine data as a secure reference. We use Algorithm 4 to implement the TPCR algorithm.

We first evaluate the accuracy of different algorithms. In fact, each data sample (a domain) in the dataset corresponds to a value, called *score*, from $[0; 1]$ indicating how likely the domain is malicious. We set a threshold to be $0{:}5$, and consider all domains with scores larger than $0{:}5$ as malicious, or benign otherwise. Then we can evaluate the accuracy as the percentage of test data that are classified correctly. The results are presented in Figure 7a. From the figure, we can observe that when the number of poisoned samples increases, the Ridge regression approach's accuracy drops dramatically, i.e., after 4000, i.e., 22%, of the entire training samples are poisoned. On the other hand, our TPCR approach remains an almost perfect accuracy until more than 10000, i.e., 55%, over the entire datasets are poisoned. We also evaluate the RMSE of different algorithms, and present the results in Figure 7b. We observe the same phenomenon as we study the accuracy of different algorithms. Therefore, we conclude that our approach provides more resilience than existing approaches.

We also observe in Figure 7b that TPCR's performance on RMSE is worse than the baseline when there are more than 10,000 number of rows corrupted. Notice that in such cases, the poisoning samples are over 50% of the training dataset. In this case, the TPCR algorithm will be fooled to think the poisoning samples, which constitute the majority, represent the "pristine data". In this case, as we prove in Section 3, it is impossible to recover the true space. Note that poisoning over half of the training data is also less realistic in real application scenarios.

7 RELATED WORK

Poisoning Attack. In this big data era, adversarial machine learning has drawn a lot of attention [4, 12, 19–21]. With the increasing popularity of crowdsourcing systems, poisoning attacks, where adversaries are able to manipulate the training data to fulfill their malicious goals, have raised severe security problems. Biggio et al. has pioneered the research of optimizing malicious data-driven attacks for kernel-based learning algorithms such as SVM [6]. The key technique for such optimization based attack strategy is to approximately compute implicit gradients of the solution of an optimization problem based on the first-order KKT conditions. Similar poisoning attack techniques have also been generalized to other widely used learning algorithms, such as Lasso regression [29], topic modeling [22], and autoregressive models [1]. A general algorithmic framework for generating poisoning attack instances on various machine learning models is analyzed in [23]. Recently, poisoning attack against deep neural networks has also been proposed to attack the substitute model via influence function [18]. In their work, the attack is restricted to only the last layer instead of the whole network. Therefore, how to conduct poisoning attack on deep networks in an end-to-end manner still remains open.

Robust Algorithm. Robust PCA is widely used as a statistical tool for data analysis and dimensionality reduction that is robust to i.i.d. noise [7]. However, these methods cannot deal with "malicious" corruptions, where the sophisticated adversaries can manipulate rows from the subspace of the true feature matrix. In contrast, our approach handles both noise and malicious corruption. Recently, robust learning for several learning models, such as linear and logistic regression have also been proposed [10, 14]. The limitation of these approaches is their assumption that the feature matrix is sub-Gaussian with vanishing variance, and that features are independent. Similarly, a provably robust algorithm has been proposed to remove random noise based on conditional correlation among data points [9]. Our approach, in contrast, only assumes that the true feature matrix (prior to corruption) is low rank. Yan *et al.* proposed an outlier pursuit algorithm to deal with the matrix completion problem with corruptions [31], and a similar algorithm is applied by Xu *et al.* to deal with the noisy version of feature matrix [30]. However, these methods only consider the matrix recovery problem and are not scalable. A more scalable algorithm based on the alternating minimization approach was recently proposed by Rodriguez et al. [24]; however, this method does not consider data noise or corruption. A number of heuristic techniques have also been proposed for poisoning attacks [3, 5, 25] for other problems, such as robust anomaly detection source identification.

8 CONCLUSION

This paper considers the poisoning attack for linear regression problem with dimensionality reduction. We address the problem in two steps: 1) introducing a novel robust matrix factorization method to recover the true subspace, and 2) novel robust principle component regression to prune adversarial instances based on the basis recovered in step (1). We characterize necessary and sufficient conditions for our approach to be successful in recovering the true subspace, and present a bound on expected prediction loss compared to ground truth. Experimental results suggest that the proposed approach is extremely effective, and significantly outperforms prior art.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for the helpful comments. This material is in part based upon work supported by the National Science Foundation under Grant No. TWC-1409915, CNS-1640624, IIS-1526860, IIS-1649972, and CNS-1238959, ONR under No. N00014-15-1-2621, ARO under No. W911NF-16-1-0069, NIH under No. UH2 CA203708-01 and R01HG006844, and Berkeley Deep Drive (BDD). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Scott Alfeld, Xiaojin Zhu, and Paul Barford. 2016. Data Poisoning Attacks against Autoregressive Models. In *AAAI*.
[2] Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Georgios Sakkis, Constantine D Spyropoulos, and Panagiotis Stamatopoulos. 2000. Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. *arXiv preprint cs/0009009* (2000).
[3] Mauro Barni and Benedetta Tondi. 2014. Source distinguishability under corrupted training. In *Information Forensics and Security (WIFS), 2014 IEEE International Workshop on*. IEEE, 197–202.
[4] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. 2006. Can machine learning be secure?. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM, 16–25.
[5] Battista Biggio, Igino Corona, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli. 2011. Bagging classifiers for fighting poisoning attacks in adversarial classification tasks. In *Multiple Classifier Systems*. Springer, 350–359.
[6] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines. In *ICML*.
[7] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. 2011. Robust principal component analysis? *Journal of the ACM (JACM)* 58, 3 (2011), 11.
[8] Philip K Chan and Salvatore J Stolfo. 1998. Toward Scalable Learning with Non-Uniform Class and Cost Distributions: A Case Study in Credit Card Fraud Detection.. In *KDD*, Vol. 1998. 164–168.
[9] Moses Charikar, Jacob Steinhardt, and Gregory Valiant. 2016. Learning from untrusted data. *arXiv preprint arXiv:1611.02315* (2016).
[10] Yudong Chen, Constantine Caramanis, and Shie Mannor. 2013. Robust High Dimensional Sparse Regression and Matching Pursuit. *arXiv preprint arXiv:1301.2725* (2013).
[11] Yudong Chen, Huan Xu, Constantine Caramanis, and Sujay Sanghavi. 2011. Robust Matrix Completion and Corrupted Columns. In *Proc. of ICML 11*.
[12] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. 2004. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 99–108.
[13] Carl Eckart and Gale Young. 1936. The approximation of one matrix by another of lower rank. *Psychometrika* 1, 3 (1936), 211–218.
[14] Jiashi Feng, Huan Xu, Shie Mannor, and Shuicheng Yan. 2014. Robust logistic regression and classification. In *Advances in Neural Information Processing Systems*. 253–261.
[15] Harold Hotelling. 1933. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology* 24, 6 (1933), 417.
[16] Ian Jolliffe. 2002. *Principal component analysis*. Wiley Online Library.
[17] Ian T Jolliffe. 1982. A note on the use of principal components in regression. *Applied Statistics* (1982), 300–303.
[18] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730* (2017).
[19] Bo Li and Yevgeniy Vorobeychik. 2014. Feature cross-substitution in adversarial classification. In *Advances in Neural Information Processing Systems*. 2087–2095.
[20] Bo Li and Yevgeniy Vorobeychik. 2015. Scalable Optimization of Randomized Operational Decisions in Adversarial Classification Settings.. In *AISTATS*.
[21] Daniel Lowd and Christopher Meek. 2005. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 641–647.
[22] Shike Mei and Xiaojin Zhu. 2015. The Security of Latent Dirichlet Allocation. In *AISTATS*.
[23] Shike Mei and Xiaojin Zhu. 2015. Using Machine Teaching to Identify Optimal Training-set Attacks on Machine Learners. In *AAAI*.
[24] Paul Rodriguez and Brendt Wohlberg. 2013. Fast principal component pursuit via alternating minimization. In *Image Processing (ICIP), 2013 20th IEEE International Conference on*. IEEE, 69–73.
[25] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and JD Tygar. 2009. Antidote: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 1–14.
[26] Claude E Shannon. 1949. Communication theory of secrecy systems. *Bell Labs Technical Journal* 28, 4 (1949), 656–715.
[27] Salvatore Stolfo, David W Fan, Wenke Lee, Andreas Prodromidis, and P Chan. 1997. Credit card fraud detection using meta-learning: Issues and initial results. In *AAAI-97 Workshop on Fraud Detection and Risk Management*.
[28] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. 2015. Is Feature Selection Secure against Training Data Poisoning?. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 1689–1698.
[29] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. 2015. Is Feature Selection Secure against Training Data Poisoning. In *ICML*.
[30] Huan Xu, Constantine Caramanis, and Sujay Sanghavi. 2010. Robust PCA via outlier pursuit. In *Advances in Neural Information Processing Systems*. 2496–2504.
[31] Ming Yan, Yi Yang, and Stanley Osher. 2013. Exact low-rank matrix completion from sparsely corrupted entries via adaptive outlier pursuit. *Journal of Scientific Computing* 56, 3 (2013), 433–449.

A PROOF OF THEOREMS ABOUT ROBUST MATRIX FACTORIZATION

In this appendix, we present the proofs of theorems in Section 3. Since this section does not involve **y**, we will omit **y** without loss of clarity.

A.1 Theorem 1

Proof of Theorem 1. We prove by contradiction. Assume we have a learner L_{recover}, can solve Problem 1 with probability more than 1/2. We want to show that there exist two different spaces of rank-k, and one input $\mathbf{X}$ such that $L_{\text{recover}}(\mathbf{X}_1)$ should return both two spaces with a probability > 1/2, which is impossible. In the following, we construct such two spaces. Particularly, we will discuss how adversary can manipulate the matrix.

The adversary can choose I which maximize $|I|$ such that while $\text{rank}(\mathbf{X}_\star^I) \leq k-1$. We know $|I| = MS_{k-1}(\mathbf{X}_\star) \geq n - n_1$. This means that $|O| - |I| \leq n_1$.

Suppose $v_1; :::; v_{k-1}$ be a set of basis for the row space of $\mathbf{X}_\star^I$. The adversary then choose a vector v'_k which is orthogonal to $\mathbf{X}_\star$. Then we know the span space of $V' = \{v_1; :::; v_{k_1}; v'_k\}$ is different from $\mathbf{X}_\star$. Then the adversary draws n_1 samples from the span space of V', and insert them into $\mathbf{X}_\star$ to form $\mathbf{X}$. Moreover, we denote $\mathbf{X}'_\star$ to be a matrix of $|I| + n_1$ rows, so that the first $|I|$ rows are $\mathbf{X}_\star^I$, and the rest n_1 rows sampled by the adversary. Therefore, we know $\mathbf{X}'_\star$ is also a submatrix of $\mathbf{X}$, and we know that there are at most $|O| - |I| \leq n_1$ rows in $\mathbf{X}$ not coming from $\mathbf{X}'_\star$.

In doing so, we know that $\mathbf{X}$ is constructed by corrupting $\mathbf{X}_\star$. On the other hand, we can also see $\mathbf{X}$ as the result of corrupting $\mathbf{X}'_\star$ by inserting $|O| - |I| \leq n_1$ rows. Therefore, $L_{\text{recover}}(\mathbf{X}_\star)$ should return both $\mathbf{X}_\star$ and $\mathbf{X}'_\star$ with a probability greater than 1/2, which is impossible. Therefore, our conclusion holds true.

A.2 Theorem 2

Proof of Theorem 2. We show that Algorithm 1 recovers the subspace of $\mathbf{X}_\star$ exactly. Assume $\mathbf{B}$ is returned by Algorithm 1 over $\mathbf{X}$. We only need to show that $\mathbf{B}$ is a basis of $\mathbf{X}_\star$. By Algorithm 1, we know that $\mathbf{B}$ is a basis of n rows in I of $\mathbf{X}$. Since we know any adversary can corrupt at most n_1 rows, thus $|I \cap A| \leq n_1$. Therefore, by combining the assumption $n_1 + MS_{k-1}(\mathbf{X}_\star) < n$, we know that

$$|I \cap O| = |I| - |I \cap A| \geq n - n_1 > MS_{k-1}(\mathbf{X}_\star) \qquad (7)$$

Therefore, we know $\mathbf{B}$ is a basis for $\mathbf{X}_\star^{I \cap O}$. By the definition of $MS_{k-1}(\mathbf{X}_\star)$ and inequality (7), we know that

$$\text{rank}(\mathbf{X}_\star^{I \cap O}) = k$$

Therefore, we know that $\mathbf{X}_\star^{I \cap O}$ is exactly the same subspace as $\mathbf{X}_\star$, and thus $\mathbf{B}$ is the basis of $\mathbf{X}_\star$.

A.3 Corollary 1

Lemma 2. $MS_{k-1}(X_\star) \geq k-1$

Proof. We can choose the $I = \{1; :::; k-1\}$, then we have $\text{rank}(\mathbf{X}_\star^I) \leq k-1$. Therefore, $MS_{k-1}(\mathbf{X}_\star) \geq |I| = k-1$.

Now, we can prove Corollary 1.

Proof of Corollary 1. Given $\frac{n_1}{n} = \gamma \geq 1 - \frac{k-1}{n}$, we have

$$n_1 + (k-1) \geq n$$

Combining $MS_{k-1}(\mathbf{X}_\star) \geq k-1$, we know

$$n_1 + MS_{k-1}(\mathbf{X}_\star) \geq n_1 + (k-1) \geq n$$

Applying Theorem 1, we can conclude this corollary.

A.4 Theorem 3

Proof of Theorem 3. The proof of this theorem is similar to the proof of Theorem 1 and 2. First, when $SR(\mathbf{X}_0) \leq NR(\mathbf{X}_0)$, the adversary can construct $\mathbf{X}$ such that two subspaces should be recovered with a probability greater than 1/2. Particularly, we assume $I; \mathbf{U}; \mathbf{B}$ minimize objective 2, and thus $SR(\mathbf{X}_0) = ||\mathbf{X}_0^I - \mathbf{UB}||$. The adversary samples n_1 rows $\mathbf{X}_{\text{corrupt}}$ from the span space of $\mathbf{B}$, which does not belong to the span of $\mathbf{X}_\star$. We add a small noise over $\mathbf{X}_{\text{corrupt}}$ to get $\mathbf{X}_1$, such that (1) $\mathbf{X}_{\text{corrupt}}$ minimize $||\mathbf{X}_1 - \mathbf{X}_{\text{corrupt}}||$; and (2) $||\mathbf{X}_1 - \mathbf{X}_{\text{corrupt}}|| = NR(\mathbf{X}_0) - SR(\mathbf{X}_0)$. Then the adversary insert $\mathbf{X}_1$ into $\mathbf{X}_0$ to get $\mathbf{X}$. In this case, we know that $\mathbf{X}_\star$ optimizes its distance from $\mathbf{X}_0$, while the $[\mathbf{X}_\star^I; \mathbf{X}_{\text{corrupt}}]$ optimizes its distance from $[\mathbf{X}_0^I; \mathbf{X}_{\text{corrupt}}]$, where we use $[A; B]$ to denote the concatenation of rows from A and B respectively. Further, by definition, we know both of these two distances is $NR(\mathbf{X}_0)$. Therefore, the learner should recover from $\mathbf{X}$ both $\mathbf{X}_\star$ and $[\mathbf{X}_0^I; \mathbf{X}_{\text{corrupt}}]$ with probability greater than 1/2. This is impossible! Therefore the first part of the theorem holds true.

For the second part, we follow the proof of Theorem 2 verbatim, and present the difference. We show that Algorithm 2 recovers the subspace of $\mathbf{X}_\star$ exactly. Assume $\mathbf{B}$ is returned by Algorithm 2 over $\mathbf{X}$. We only need to show that $\mathbf{B}$ is a basis of $\mathbf{X}_\star$. By Algorithm 2, we know that $\mathbf{B}$ optimizes its pan distance from a subset of n rows in $\mathbf{X}$, which is denoted as I. Since we know any adversary can corrupt at most n_1 rows, thus $|I \cap A| \leq n_1$. Therefore, we know that

$$|I \cap O| = |I| - |I \cap A| \geq n - n_1 \qquad (8)$$

If $\mathbf{B}$ is not a basis of $\mathbf{X}_\star$, which means that $\mathbf{X}_\star \mathbf{B}^T \mathbf{B} \neq \mathbf{X}_\star$, then we know that the distance between the span space of $\mathbf{B}$ and $\mathbf{X}^{I \cap O}$ is greater than $SR(\mathbf{X}_0) > NR(\mathbf{X}_0)$. This is impossible, since Algorithm 2 guarantees that this distance should be no greater than $NR(\mathbf{X}_0)$. Contradiction! Therefore the second part of the theorem holds true.

A.5 Theorem 4

Proof of Theorem 4. When $\mathbf{N} = 0$, we know that $SR(\mathbf{X}_0) > NR(\mathbf{X}_0)$ if and only if $SR(\mathbf{X}_0) \neq 0$. This means that for any $|I| = n - n_1$, $\mathbf{X}_\star^I = \mathbf{UB}$ implies that $\mathbf{X}_\star \mathbf{B}^T \mathbf{B} = \mathbf{X}_\star$ (condition (2b)), which implies that $\text{rank}(\mathbf{X}_\star^I |) = k$ for all I. Therefore, we know $MS_{k-1}(\mathbf{X}_\star) < n - n_1$, which concludes this theorem.

B PROOF OF TPCR LEMMA

We present the proof of TPCR Lemma 1 below.

Proof. Assume $\widehat{\beta_U}$ is the solution for this optimization problem. We assume the adversary wants to induce the regression system to compute $\widehat{\beta_U}$. In this case, he has to corrupt γn rows in U. W.L.O.G. we can assume $O = \{1, \ldots, n_1\}$. We denote $\beta_U^\star = \mathbf{B}\beta^\star$. Since $\mathbf{X}_\star = \mathbf{U}_\star\mathbf{B}$, we know that

$$\mathbf{y} - \mathbf{X}_\star\beta^\star = \mathbf{y} - \mathbf{U}_\star\beta_U^\star$$

Since $\widehat{\beta_U}$ optimize Eq (4), we assume $(y_i - u_i\widehat{\beta_U})^2$ are the smallest n values for $i \in \{1, \ldots, n\}$.

Then we have

$$\sum_{i=1}^{n_1}(y_i - u_i\widehat{\beta_U})^2 + \sum_{i=n_1+1}^{n}(y_i - u_i\widehat{\beta_U})^2 \le \sum_{i=n_1+1}^{n+n_1}(y_i - u_i^T\beta_U^\star)^2$$

Therefore we have

$$\sum_{i=n_1+1}^{n}(y_i - u_i\widehat{\beta_U})^2 \le \sum_{i=n_1+1}^{n+n_1}(y_i - u_i\beta_U^\star)^2 \qquad (9)$$

Further, we know

$$\begin{aligned}
&\sum_{i=n_1+1}^{n}(y_i - u_i\widehat{\beta_U})^2 \\
= &\sum_{i=n_1+1}^{n}\left((y_i - u_i\beta_U^\star) + (u_i\beta_U^\star - u_i\widehat{\beta_U})\right)^2 \\
\ge &\sum_{i=n_1+1}^{n}(y_i - u_i\beta_U^\star)^2 + (u_i\beta_U^\star - u_i\widehat{\beta_U})^2 \\
&\quad -2|y_i - u_i\beta_U^\star| \cdot |u_i\beta_U^\star - u_i\widehat{\beta_U}| \\
= &\sum_{i=n_1+1}^{n}(y_i - u_i\beta_U^\star)^2 + \sum_{i=n_1+1}^{n}(u_i(\beta_U^\star - \widehat{\beta_U}))^2 \\
&\quad -2\sum_{i=n_1+1}^{n}|u_i(\beta_U^\star - \widehat{\beta_U})| \cdot |y_i - u_i\beta_U^\star| \qquad (10)
\end{aligned}$$

According to Cauchy-Schwarz inequality, we have

$$\begin{aligned}
&\left(\sum_{i=n_1+1}^{n}|u_i(\beta_U^\star - \widehat{\beta_U})| \cdot |y_i - u_i\beta_U^\star|\right)^2 \\
\le &\sum_{i=n_1+1}^{n}(u_i(\beta_U^\star - \widehat{\beta_U}))^2 \cdot \sum_{i=n_1+1}^{n}(y_i - u_i\beta_U^\star)^2
\end{aligned}$$

We assume $C = \sqrt{\sum_{i=n_1+1}^{n}(u_i(\beta_U^\star - \widehat{\beta_U}))^2}$, then, we have

$$\begin{aligned}
&-2\sum_{i=n_1+1}^{n}|u_i(\beta_U^\star - \widehat{\beta_U})| \cdot |y_i - u_i\beta_U^\star| \\
\ge &-2\sqrt{\sum_{i=n_1+1}^{n}(u_i(\beta_U^\star - \widehat{\beta_U}))^2 \cdot \sum_{i=n_1+1}^{n}(y_i - u_i\beta_U^\star)^2} \\
= &-2C\sqrt{\Sigma_{i=n_1+1}^{n}e_i^2}
\end{aligned}$$

Substituting this inequality into (10) and combining with (9), we have

$$\sum_{i=n_1+1}^{n+n_1}e_i^2 \ge \sum_{i=n_1+1}^{n}e_i^2 + C^2 - 2C\sqrt{\Sigma_{i=n_1+1}^{n}e_i^2}$$

By simple rearrangement, we have

$$C^2 - 2C\sqrt{\sum_{i=n_1+1}^{n}e_i^2} \le \sum_{i=n+1}^{n+n_1}e_i^2$$

Since we know $y_i - u_i\beta_U^\star \sim \mathcal{N}(0, \sigma)$, we know that for any parameter $h > 1$, we have $\Pr(e_i \le 2\sigma\sqrt{\log h}) \ge 1 - ch^{-2}$ for some constant c. Therefore, we know, with high probability (at least $1 - ch^{-2}$), we have

$$\begin{aligned}
C^2 - 2\sqrt{n-n_1}C(2\sigma\sqrt{\log h}) &\le C^2 - 2C\sqrt{\sum_{i=n+1}^{n}e_i^2} \\
&\le \sum_{i=n+1}^{n+n_1}e_i^2 \\
&\le n_1(2\sigma\sqrt{\log h})^2
\end{aligned}$$

Therefore, we have

$$\left(C - 2\sigma\sqrt{n-n_1}\sqrt{\log h}\right)^2 \le n(2\sigma\sqrt{\log h})^2$$

and thus

$$C \le 2\sigma\left(\sqrt{n} + \sqrt{n-n_1}\right)\sqrt{\log h}$$

Therefore, we know

$$\sqrt{\frac{\sum_{i=n_1}^{n}||u_i(\beta_U^\star - \widehat{\beta_U})||^2}{n-n_1}} \le 2\sigma\left(1 + \sqrt{\frac{1}{1-\gamma}}\right)\sqrt{\log h}$$

We notice the right hand side of the above inequality does not depend on n, n_1. Therefore, we take $n \longrightarrow +\infty$, and we know that $n - n_1 = (1-\gamma)n \longrightarrow +\infty$, and apply the law of large numbers, we have

$$\sqrt{E_u\left(u(\beta_U^\star - \widehat{\beta_U})\right)^2} \le 2\sigma\left(1 + \sqrt{\frac{1}{1-\gamma}}\right)\sqrt{\log h}$$

where left hand side is the same as $\sqrt{E_x\left(x(\widehat{\beta} - \beta^\star)\right)^2}$]. Then the conclusion of Lemma 1 is a simple rearrangement of the above inequality.