Robust Linear Regression Against Training Data Poisoning

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

Yevgeniy Vorobeychik
Vanderbilt University
yevgeniy.vorobeychik@vanderbilt.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 \(y\) are a linear function of the true feature matrix \(X^\star\) with additive zero-mean noise. In addition,
$X_\star$ 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 sub-
space 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 regres-
sion 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 rela-
tion between $X_i$ and $y_i$, i.e., $f(X_i)$ and $y_i$ are close. Depending
on the values $y_i$ 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 $(X, y)$, where $y$
is the dependent variable depending on $X$, a linear model assumes
that $y$ is “close” to $\beta \cdot X$ with respect to the distance function, for
a set of parameters $\beta$.

In this section, we first present the threat model; then we formal-
ize 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 at-
tackers, 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 Stack-
elberg 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,
$(X_\star, y_\star)$, which subsequently suffers from two types of corrup-
tion: 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 dimension-
ality reduction prior to learning. In particular, we assume that $X_\star$
is low-rank with a basis $B$, 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**: $y_\star = X_\star \beta^\star = U \beta^\star_2$, where $\beta^\star$ is the
   true model, $\beta^\star_2$ is its low-dimensional representation, and
   $U = X_\star B$ is the low-dimensional embedding of $X_\star$
2. **Noise**: $X_0 = X_\star + N$, where $N$ is a noise matrix with
   $\|N\|_\infty \leq \epsilon$; $y_0 = y_\star + \epsilon$, where $\epsilon$ is i.i.d. zero-mean Gaussian
   noise with variance $\sigma$.
3. **Corruption**: The attacker adds $n_1$ adversarially crafted
   examples $(x_a, y_a)$ to get $(X, y)$, which maximally skews
   prediction performance of low-dimensional linear regres-

To formally characterize how well the learner performs in this
setting, we define (1) a *model function* $f(X_0, y_0)$ which is the model
learned on $(X_\star, y_\star)$; (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 $\mathcal{L}$ is
$(f, l, \delta)$-tolerant, if for any attacker, and any $z > 1$, we have
$l(\mathcal{L}(X, y), f(X_0, y_0)) \leq \delta(z)$
with probability at least $1 - c_1 z^{-c_2}$, for some constant $c_1, c_2 > 0$.

In our setting, $f(X_0, y_0)$ returns $\beta^\star$ and $l$ is expected quadratic
loss $E_\star [\ell(x(\hat{\beta} - \beta^\star))^2]$. 

### Notation Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>Number of pristine samples</td>
</tr>
<tr>
<td>$n_1$</td>
<td>Number of poisoning samples</td>
</tr>
<tr>
<td>$n + n_1$</td>
<td>Total number of training samples</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>The corruption ratio $n_1/n$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Estimated parameter vector</td>
</tr>
<tr>
<td>$\beta^*$</td>
<td>True parameter vector</td>
</tr>
<tr>
<td>$\beta^*_L$</td>
<td>Low-dimensional representation of $\beta^*$</td>
</tr>
<tr>
<td>$(X_0, y_0)$</td>
<td>Pristine training data</td>
</tr>
<tr>
<td>$(X_\star, y_\star)$</td>
<td>Examples with Gaussian uncertainty</td>
</tr>
<tr>
<td>$(\tilde{X}, \tilde{y})$</td>
<td>Adversarially crafted examples</td>
</tr>
<tr>
<td>$(\tilde{X}, \tilde{y})$</td>
<td>Dataset after injecting poisoning instances</td>
</tr>
<tr>
<td>$B$</td>
<td>Basis for $X_\star$</td>
</tr>
<tr>
<td>$f$</td>
<td>Learning model</td>
</tr>
<tr>
<td>$l$</td>
<td>Loss function</td>
</tr>
</tbody>
</table>

For convenience, we let $\mathcal{O}$ denote the set of (unknown) indices of the samples in $X$ coming from $X_0$ and $\mathcal{O} = \{1, \ldots, n + n_1\} - \mathcal{O}$ the set of indices for adversarial samples in $X$. For an index set $\mathcal{I}$ and matrix $M, M^{\mathcal{I}}$ denotes the sub-matrix containing only rows in $\mathcal{I}$; similar notation is used for vectors. We define $\gamma = n_1/n$ 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.

### 3. ROBUST SUBSPACE RECOVERY

In this section, we discuss how to recover the low-rank subspace of $X_\star$ from $X$. Our goal is to exactly recover the low-rank subspace, i.e., returning a basis for $X_\star$. 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 $\mathcal{L}_{\text{recovery}}$, 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 $X^\mathcal{O} = X_\star$. We assume that we know $\text{rank}(X_\star) = k$ (or have an upper bound on it). Below we demonstrate that there exists a sharp threshold $\theta$ 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 $\text{MS}_{k-1}(X_\star)$ as the optimal value of the following problem:

$$\max_{|\mathcal{I}|} |\mathcal{I}| \text{ s.t. } \text{rank}(X_{\mathcal{I}}^\mathcal{O}) \leq k - 1$$

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

**Theorem 1.** If $n_1 + \text{MS}_{k-1}(X_\star) \geq n$, then there exists an adversary such that no algorithm $\mathcal{L}_{\text{recovery}}$ 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 $X_\star$.

**Algorithm 1** Exact recover algorithm for Problem 1 (Noisy-free)

We search for a subset $\mathcal{I}$ of indices, such that $|\mathcal{I}| = n$, and $\text{rank}(X_{\mathcal{I}}^\mathcal{O}) = k$.

return a basis of $X_{\mathcal{I}}^\mathcal{O}$.

In fact, we can prove the following theorem.

**Theorem 2.** If $n_1 + \text{MS}_{k-1}(X_\star) < n$, 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 $\gamma$. We can prove that $\text{MS}_{k-1}(X_\star) \geq k - 1$ (see Appendix A). Combining this with Theorem 1, we obtain the following upper bound on $\gamma$.

**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 algorithm, 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:

\[
\begin{align*}
\min_{X'} ||X_0 - X'|| & \quad (1a) \\
\text{s.t.} \, \text{rank}(X') \leq k. & \quad (1b)
\end{align*}
\]

Unless otherwise mentioned, we use $|| \cdot ||$ to denote the Frobenius norm. We put no additional restrictions on $N$ 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 $NR(X_0) = N$. 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 $B$ of $X_\star$ after the attacker adds $n_1$ malicious instances stems from the attacker’s ability to mislead the defender into thinking that some other basis, $B$, better represents $X_\star$. Intuitively, since the defender does not know $X_0$, $X_\star$, or which $n_1$ rows of the data matrix $X$ 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 all adversarial rows are used, since the adversary may be forced to align malicious samples 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 $NR(X_0)$ and sub-matrix residual, denoted as $SR(X_0)$, which we define as the value optimizing the following problem:

\[
\begin{align*}
\min_{\mathcal{F},B,U} ||X_0^\mathcal{F} - UB|| & \quad (2a) \\
\text{s.t.} \text{rank}(B) = k, BB^T = I_k, X_\star B^T \neq X_\star & \quad (2b) \\
\mathcal{F} \subseteq \{1, 2, ..., n\}, |\mathcal{F}| = n - n_1, & \quad (2c)
\end{align*}
\]

We now explain the above optimization problem. $U$ and $B$ are $(n - n_1) \times k$ and $k \times m$ matrices separately. Here $B$ is a basis which the attacker “targets”; for convenience, we require $B$ to be orthogonal (i.e., $BB^T = I_k$). Since the attacker succeeds only if they can induce a basis different from the true $B$, we require that $B$ does not span of $X_\star$, which is equivalent to saying $X_\star B^T B \neq X_\star$. Thus, this optimization problem seeks $n - n_1$ rows of $X_\star$, where $\mathcal{F}$ is the corresponding index set. The objective is to minimize the distance between $X_0^\mathcal{F}$ and the span space of the target basis $B$, (i.e., $||X_0^\mathcal{F} - UB||$).

\[\text{Algorithm 2} \text{ Exact recovery algorithm for Problem 1} \]
\begin{algorithmic}
\STATE Solve the following optimization problem and get $\mathcal{F}$. 
\STATE \vspace{0.2cm}
\begin{align*}
\min_{\mathcal{F},L} ||X^\mathcal{F} - L|| & \quad (3a) \\
\text{s.t.} \text{rank}(L) \leq k, \mathcal{F} \subseteq \{1, 2, ..., n + n_1\}, |\mathcal{F}| = n & \quad (3b)
\end{align*}
\STATE return a basis of $X^\mathcal{F}$. 
\end{algorithmic}

To understand the importance of $SR(X_0)$, consider Algorithm 2 for recovering the basis of $X_\star$, $B$. If the optimal objective value of optimization problem (2), $SR(X_0)$, exceeds the noise $NR(X_0)$, it follows that the defender can obtain the correct basis $B$ using Algorithm 2, as it yields a better low-rank approximation of $X$ 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 $\beta$, 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 $X_\star$.

Assume we recover a basis $B$ of $X_\star$. Without loss of generality, we assume that $B$ is an orthogonal basis of $k$ row vectors. Since $B$ is a basis for $X_\star$, we assume $X_\star = UB$. Then we know that, by optimization (1), $U_\star = \text{argmin}_U(||X_0 - UB||)$. We compute $U = \text{argmin}_U(||X - UB||)$, and, by definition, we know $U_\star = U^\mathcal{F}$. By OLS estimator, we know that $U^T = (BB^T)^{-1}BX^T$, and thus $U = XB^T$.

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

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

\[\text{Algorithm 3} \text{ Trimmed Principal Component Regression} \]
\begin{algorithmic}
\STATE (1) Use Algorithm 2 to compute a basis from $X$, and orthogonalize it to get $B$
\STATE (2) Project $X$ onto the span space of $B$ and get $U \leftarrow XB^T$
\STATE (3) Solve the following minimization problem to get $\beta_U$
\STATE \vspace{0.2cm}
\begin{align*}
\min_{\beta_U} \sum_{i=1}^{n_1} \left( y_i - u_i \beta_U \right)^2 \text{ for } i = 1, ..., n + n_1 \right)_{(i)} & \quad (4)
\end{align*}
\STATE \vspace{0.2cm}
where $z_{(j)}$ denotes the $j$-th smallest element in sequence $z$.
\STATE (4) return $\hat{\beta} \leftarrow B\hat{\beta}_U$.
\end{algorithmic}
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 $u_i \hat{b}_i$, 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, $\sigma$ is the variance of the random noise $y - x\beta^*$), 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 $X_\ast, y_\ast$.

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

$$
E_x[(x - \beta)^2] \leq 4\sigma^2 \left(1 + \frac{1}{1 - \gamma}\right)^2 \log c,
$$

(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 $(y_i - y_i \hat{\beta}_i)^2$ is among the smallest $n$. Since for samples from $X_\ast, Y_\ast$, these loss functions are already bounded according to $\sigma$, 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 $y_\ast$ from $X_\ast$, 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 $\delta(c) = 4\sigma^2 \left(1 + \frac{1}{1 - \gamma}\right)^2 \log c$, Algorithm 3 is $(f, 1, \delta(c))$-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 rank$(L) \leq k$, we can rewrite $L = UB^T$ where $U$‘s and $B$’s shapes are $n \times k$, and $m \times k$ respectively. Therefore, we can rewrite objective (3) as

$$
\min_{\mathcal{F}, U, B} \|X - UB^T\| \text{ s.t. } |\mathcal{F}| = n
$$

which is equivalent to

$$
\min_{U, B} \sum_{i=1}^{n} \|x_i - u_i B^T\| \text{ for } i = 1, \ldots, n + n_1 \langle j \rangle
$$

(6)

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

$$
U^{w+1} = \arg\min_U \|X - U(B^w)^T\|
$$

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 $\theta \leftarrow \arg\min_{\theta} \sum_{i=1}^{n + n_1} \tau_i l(y_i, f_\theta(x_i))$
3. Compute rank $\gamma$ as the rank of $l(y_i, f_\theta(x))$ in the ascending order.
4. Set $\tau_i \leftarrow 1$ for $\gamma_i \leq n$, and $\tau_i \leftarrow 0$ otherwise.
5. Go to 2 if any of $\tau_i$ changes;
6. **Return** $\theta$.

$$
B^{w+1} = \arg\min_B \sum_{j=1}^{n} \|x_i - u_i^{w+1} B^T\| \text{ for } i = 1, \ldots, n + n_1 \langle j \rangle
$$

Notice that the second step computes the entire $U$ regardless of the sub-matrix restriction. This is because we need the entire $U$ to be computed to update $B$. The key challenge is to compute $B^{w+1}$ 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

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

where $f_\theta(x_i)$ computes the prediction over $x_i$ using parameter $\theta$, 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

$$
\min_{\theta, \tau_1, \ldots, \tau_n, n_1} \sum_{i=1}^{n + n_1} \tau_i l(y_i, f_\theta(x_i)) \text{ s.t. } 0 \leq \tau_i \leq 1, \sum_{i=1}^{n + n_1} \tau_i = n
$$

We can use alternating minimization technique to solve this problem, by optimizing for $\theta$, and $\tau_i$ respectively. We present this in Algorithm 4. In particular, the algorithm iteratively seeks optimal arguments for $\theta$ and $\tau_1, \ldots, \tau_{n + n_1}$ respectively. Optimizing for $\theta$ 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 $\tau$ 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 an 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 $X_\star$ as follows: sample two matrices $U, B$ 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., $U$ or $B$) 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 $U$ and $B$ are sampled, and we set $X_\star = UB$. We do not add noise to $X_\star$, unless explicitly stated. When noise is added, the pristine data is generated as $X_\star + N$, where each element in $N$ 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 HTTP’s 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 referrer domains, fraction of connections without referrer); 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 February, 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 $X_\star$, the generated pristine data. We generate corruptions $X^{df}$ also as a low rank matrix by generating $U^{df}$ and $B^{df}$ in the same way as generating $U$ and $B$, where $U^{df}$ has $n_1$ rows.

For $B^{df}$, we set the first half of $B^{df}$ by choosing $k/2$ rows of $X_\star$, and generating the remaining $k/2$ rows randomly, while ensuring that $B$ has rank $k$. We concatenate $X_\star$ and $X^{df} = U^{df}B^{df}$ to get the matrix with $n + n_1$ rows, and then shuffle them. In doing so, we know that $X^{df}$ shares a common subspace of rank $k/2$ with $X_\star$, 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
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 dominates 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 approach’s when $k$ is increased.

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 dominates 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 $\gamma$ 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].
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](image1.png) and ![Figure 4b](image2.png) show the RMSE of the difference from recovered $X^G$ and the true $X^\star$. 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 $SR(X_0) > NR(X_0)$ 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.

![Figure 5](image3.png) shows the runtime comparison results. 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.

![Figure 6](image4.png) shows the RMSE comparison for robust regression. When compared with vanilla linear regression, both of our approach and Chen et al. 2013 are around $7 \times$ 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
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 (February, 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**

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 

A.1 Theorem 1

Proof of Theorem 1. We prove by contradiction. Assume we have a learner \( L_{\text{recover}} \), 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 \( X \) such that \( L_{\text{recover}}(X_1) \) should return both 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 \( \mathcal{I} \) which maximize \( \| \mathcal{A} \| \) such that while \( \text{rank}(X_{\mathcal{A}}^\star) \leq k - 1 \). We know \( \| \mathcal{A} \| = MS_{k-1}(X^\star) \geq n - n_1 \). This means that \( \| \mathcal{A} \| - \| \mathcal{A} \| \leq n_1 \).

Suppose \( v_1, ..., v_{k-1} \) be a set of basis for the row space of \( X \). The adversary then choose a vector \( v_k \) which is orthogonal to \( X \). Then we know the span space of \( V' = \{v_1, ..., v_{k-1}, v_k\} \) is different from \( X \). Then the adversary draws \( n_1 \) samples from the span space of \( V' \), and insert them into \( X \) to form \( X' \). Moreover, we denote \( X' \) as the result of corrupting \( X \), and know that there are at most \( \| \mathcal{A} \| - \| \mathcal{A} \| \leq n_1 \) rows in \( X' \) not coming from \( X_{\mathcal{A}} \).

In doing so, we know that \( X \) is constructed by corrupting \( X \), by inserting \( v_k \) which is orthogonal to \( X \) from \( X \). Then we know \( \mathcal{I} \) returns both \( X \) and \( X' \) 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 \( X \) exactly. Assume \( B \) is returned by Algorithm 1 over \( X \). We only need to show that \( B \) is a basis of \( X \). By Algorithm 1, we know that \( B \) is a basis of \( n \) rows in \( \mathcal{I} \). Since we know any adversary can corrupt at most \( n_1 \) rows, thus \( \mathcal{I} \cap \mathcal{A} \leq n_1 \). Therefore, by combining the assumption \( n_1 + MS_{k-1}(X^\star) < n \), we know that

\[
\| \mathcal{A} \| = \| \mathcal{A} \| - \| \mathcal{A} \| \geq n - n_1 > MS_{k-1}(X^\star)
\]

Therefore, we know \( B \) is a basis for \( X_{\mathcal{A}} \). By the definition of \( MS_{k-1}(X^\star) \) and inequality (7), we know that

\[
\text{rank}(X_{\mathcal{A}}) = k
\]

Therefore, we know that \( X_{\mathcal{A}} \) is exactly the same subspace as \( X \), and thus \( B \) is the basis of \( X \).

A.3 Corollary 1

Lemma 2. \( MS_{k-1}(X^\star) \geq k - 1 \)

Proof. We can choose the \( \mathcal{I} = \{1, ..., k - 1\} \), then we have \( \text{rank}(X_{\mathcal{A}}) \leq k - 1 \). Therefore, \( MS_{k-1}(X^\star) \geq |\mathcal{I}| = k - 1 \).

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 \( MS_{k-1}(X^\star) \geq k - 1 \), we know \( n_1 + MS_{k-1}(X^\star) \geq n + (k-1) \geq n \)

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 \( SR(X_0) \leq NR(X_0) \), the adversary can construct \( X \) such that two subspaces should be recovered with a probability greater than 1/2. Particularly, we assume \( \mathcal{I}, U, B \) minimize objective 2, and thus \( SR(X_0) = \| X_{\mathcal{A}}^\star - UB \| \).

The adversary samples \( n_1 \) rows \( X_{\mathcal{A}} \) from the span space of \( B \), which does not belong to the span of \( X^\star \). We add a small noise over \( X_{\mathcal{A}} \) to get \( X \), such that (1) \( X_{\mathcal{A}} \) minimize \( \| X_1 - X_{\mathcal{A}} \| \), and (2) \( \| X_1 - X_{\mathcal{A}} \| = SR(X_0) - SR(X_0) \). Then the adversary corrupt \( X_1 \) into \( X_0 \) to get \( X \). In this case, we know that \( X^\star \) optimizes its distance from \( X_0 \), while \( \{X_{\mathcal{A}} \} \) optimizes its distance from \( \{X_{\mathcal{A}} \}^\star \). In this case, 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 \( SR(X_0) \). Therefore, the learner should recover \( X^\star \) from both \( X_0 \) and \( \{X_{\mathcal{A}} \}^\star \) 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 \( X_{\mathcal{A}} \) exactly. Assume \( B \) is returned by Algorithm 2 over \( X \). We only need to show that \( B \) is a basis of \( X \). By Algorithm 2, we know that \( B \) optimizes its pan distance from a subset of \( n \) rows in \( X \), which is denoted as \( \mathcal{I} \). Since we know any adversary can corrupt at most \( n_1 \) rows, thus \( \mathcal{I} \cap \mathcal{A} \leq n_1 \). Therefore, we know that

\[
\| \mathcal{A} \| = \| \mathcal{A} \| - \| \mathcal{A} \| \geq n - n_1 > MS_{k-1}(X^\star)
\]

If \( B \) is not a basis of \( X \), which means that \( X_{\mathcal{A}}^\star B \neq X_{\mathcal{A}} \), then we know that the distance between the span space of \( B \) and \( X_{\mathcal{A}} \) is greater than \( NR(X_0) \). This is impossible, since Algorithm 2 guarantees that this distance should be no greater than \( NR(X_0) \). Contradiction! Therefore the second part of the theorem holds true.

A.5 Theorem 4

Proof of Theorem 4. When \( N = 0 \), we know that \( SR(X_0) > NR(X_0) \) if and only if \( SR(X_0) \neq 0 \). This means that for any \( \| \mathcal{I} \| = n - n_1 \), \( X_{\mathcal{A}} = UB \) implies that \( X_{\mathcal{A}}^\star B^\star = X_{\mathcal{A}} \), which implies that \( \text{rank}(X_{\mathcal{A}}) = k \) for all \( \mathcal{I} \). Therefore, we know \( MS_{k-1}(X^\star) < n - n_1 \), which concludes this theorem.
B PROOF OF TPCR LEMMA

We present the proof of TPCR Lemma 1 below.

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

$$y - X\beta^* = y - U\beta_{\text{U}}^*$$

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

Then we have

$$\sum_{i=1}^{n_1} (y_i - u_i\beta_{\text{U}})^2 + \sum_{i=n_1+1}^{n} (y_i - u_i\beta_{\text{U}})^2 \leq \sum_{i=n_1+1}^{n+n_1} (y_i - u_i\beta_{\text{U}}^*)^2$$

Therefore we have

$$\sum_{i=n_1+1}^{n+n_1} (y_i - u_i\beta_{\text{U}}^*)^2 \leq \sum_{i=n_1+1}^{n+n_1} (y_i - u_i\beta_{\text{U}})^2$$

(9)

Further, we know

$$\sum_{i=n_1+1}^{n} (y_i - u_i\beta_{\text{U}})^2 = \sum_{i=n_1+1}^{n} (u_i\beta_{\text{U}} - u_i\beta_{\text{U}})^2 \leq \sum_{i=n_1+1}^{n} (u_i\beta_{\text{U}}^* - u_i\beta_{\text{U}})^2$$

$$\geq \sum_{i=n_1+1}^{n} (u_i\beta_{\text{U}}^* - u_i\beta_{\text{U}})^2 + \sum_{i=n_1+1}^{n} (u_i\beta_{\text{U}}^* - u_i\beta_{\text{U}})^2$$

$$-2|y_i - u_i\beta_{\text{U}}^*| |y_i - u_i\beta_{\text{U}}^*|$$

$$= \sum_{i=n_1+1}^{n} (y_i - u_i\beta_{\text{U}})^2 + \sum_{i=n_1+1}^{n} (u_i(\beta_{\text{U}}^* - \beta_{\text{U}}))^2$$

$$-2\left(\sum_{i=n_1+1}^{n} |u_i(\beta_{\text{U}}^* - \beta_{\text{U}})| \cdot |y_i - u_i\beta_{\text{U}}^*|\right)$$

(10)

According to Cauchy-Schwarz inequality, we have

$$\sum_{i=n_1+1}^{n} |u_i(\beta_{\text{U}}^* - \beta_{\text{U}})| \cdot |y_i - u_i\beta_{\text{U}}^*|$$

$$\leq \left(\sum_{i=n_1+1}^{n} (u_i(\beta_{\text{U}}^* - \beta_{\text{U}}))^2\right)^{\frac{1}{2}} \cdot \left(\sum_{i=n_1+1}^{n} (y_i - u_i\beta_{\text{U}})^2\right)^{\frac{1}{2}}$$

We assume $C = \sqrt{\sum_{i=n_1+1}^{n} (u_i(\beta_{\text{U}}^* - \beta_{\text{U}}))^2}$, then, we have

$$-2\left(\sum_{i=n_1+1}^{n} |u_i(\beta_{\text{U}}^* - \beta_{\text{U}})| \cdot |y_i - u_i\beta_{\text{U}}^*|\right) \geq -2\left(\sum_{i=n_1+1}^{n} (u_i(\beta_{\text{U}}^* - \beta_{\text{U}}))^2\right) \cdot \left(\sum_{i=n_1+1}^{n} (y_i - u_i\beta_{\text{U}})^2\right)$$

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

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

$$\sum_{i=n_1+1}^{n+n_1} e_i^2 \leq n(2\sigma\sqrt{\log h})^2$$

By simple rearrangement, we have

$$C^2 - 2\sigma\sqrt{n}C(2\sigma\sqrt{\log h}) \leq C^2 - 2C\sum_{i=n_1+1}^{n} e_i^2$$

$$\leq \sum_{i=n_1+1}^{n+n_1} e_i^2$$

$$\leq n(2\sigma\sqrt{\log h})^2$$

Since we know $y_i - u_i\beta_{\text{U}}^* \sim N(0, \sigma)$, we know that for any parameter $h > 1$, we have $\Pr(e_i \leq 2\sigma\sqrt{\log h}) \geq 1 - ch^{-2}$ for some constant $c$. Therefore, we know, with high probability (at least $1 - ch^{-2}$), we have

$$C^2 - 2\sigma\sqrt{n}C(2\sigma\sqrt{\log h}) \leq C^2 - 2C\sum_{i=n_1+1}^{n} e_i^2$$

Therefore, we have

$$C \leq 2\sigma\sqrt{n + \sigma(\log h)}$$

and thus

$$C \leq 2\sigma\sqrt{n + \sigma(\log h)}$$

Therefore, we know

$$\sqrt{\sum_{i=n_1+1}^{n} (u_i(\beta_{\text{U}}^* - \beta_{\text{U}}))^2} \leq 2\sigma\sqrt{n + \sigma(\log h)}$$

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

$$\sqrt{\sum_{i=n_1+1}^{n} (u_i(\beta_{\text{U}}^* - \beta_{\text{U}}))^2} \leq 2\sigma\sqrt{n + \frac{1}{1-\gamma}\sqrt{\log h}}$$

where left hand side is the same as $\sqrt{E_\sigma[(x(\beta - \beta^*))^2]}$. Then the conclusion of Lemma 1 is a simple rearrangement of the above inequality. \qed