Stronger Data Poisoning Attacks Break Data Sanitization Defenses

Pang Wei Koh*

PANGWEI@CS.STANFORD.EDU

Department of Computer Science Stanford University Stanford, CA 94305, USA

Jacob Steinhardt*

JSTEINHARDT@CS.STANFORD.EDU

Department of Computer Science Stanford University Stanford, CA 94305, USA

Percy Liang

PLIANG@CS.STANFORD.EDU

Department of Computer Science Stanford University Stanford, CA 94305, USA

Abstract

Machine learning models trained on data from the outside world can be corrupted by data poisoning attacks that inject malicious points into the models' training sets. A common defense against these attacks is data sanitization: first filter out anomalous training points before training the model. Can data poisoning attacks break data sanitization defenses? In this paper, we develop three new attacks that can all bypass a broad range of data sanitization defenses, including commonly-used anomaly detectors based on nearest neighbors, training loss, and singular-value decomposition. For example, our attacks successfully increase the test error on the Enron spam detection dataset from 3% to 24% and on the IMDB sentiment classification dataset from 12% to 29% by adding just 3% poisoned data. In contrast, many existing attacks from the literature do not explicitly consider defenses, and we show that those attacks are ineffective in the presence of the defenses we consider. Our attacks are based on two ideas: (i) we coordinate our attacks to place poisoned points near one another, which fools some anomaly detectors, and (ii) we formulate each attack as a constrained optimization problem, with constraints designed to ensure that the poisoned points evade detection. While this optimization involves solving an expensive bilevel problem, we explore and develop three efficient approximations to this problem based on influence functions; minimax duality; and the Karush-Kuhn-Tucker (KKT) conditions. Our results underscore the urgent need to develop more sophisticated and robust defenses against data poisoning attacks.

Keywords: data poisoning, data sanitization, anomaly detection, security

1. Introduction

In high-stakes settings like autonomous driving (Gu et al., 2017), biometrics (Chen et al., 2017), and cybersecurity (Rubinstein et al., 2009; Suciu et al., 2018), it is crucial that

©2018 Pang Wei Koh, Jacob Steinhardt, and Percy Liang.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/.

^{*.} These authors contributed equally.

machine learning (ML) systems be secure against attacks by malicious actors. Securing ML systems is complicated by the fact that they are often trained on data obtained from the outside world, which makes them especially vulnerable: by attacking this data collection process, which could be as easy as creating a new user account, adversaries can inject malicious data into the system and cause it to fail.

These data poisoning attacks are the focus of the present work. We consider attacks against classifiers, wherein an attacker adds some small fraction of new training points to degrade the performance of the trained classifier on a test set. Figure 1 illustrates this setting: a model that might otherwise correctly classify most of the data (Figure 1-Left) can be made to learn a significantly different decision boundary by an attacker who injects just a small amount of poisoned data (Figure 1-Middle).

A common defense against data poisoning attacks are data sanitization defenses (Cretu et al., 2008), which use anomaly detectors to filter out training points that look suspicious (see, e.g., Hodge and Austin (2004) for a review). Others have developed data sanitization defenses that are effective against some data poisoning attacks (e.g., Paudice et al. (2018)). Figure 1-Right illustrates a hypothetical defense: the poisoned data is clearly anomalous, and by removing it, the defender can learn the correct decision boundary. The success of data sanitization in this case is not surprising, as many data poisoning attacks are developed without explicitly considering defenses. Instead, attackers might for instance be given an attack budget (Mei and Zhu, 2015b) or constrained to add points that belong to the input domain (e.g., word counts in a document should be integer-valued (Nelson et al., 2008; Newell et al., 2014)). Indeed, none of the existing data poisoning methods that we tested in our experiments were able to evade the data sanitization defenses we considered.

Previous work has suggested that attacks optimized for evading data sanitization can in fact evade some types of defenses (Steinhardt et al., 2017). This observation leads to a natural question that has yet to be systematically studied: can data poisoning attacks that are explicitly designed to evade anomaly detection get around data sanitization defenses? Can defenders who deploy data sanitization still be vulnerable to attack?

In this paper, we answer those questions in the affirmative. Our key contribution is to develop three new data poisoning attacks that can simultaneously evade a broad range of data sanitization defenses, including commonly-used anomaly detectors based on nearest neighbors, training loss, singular-value decomposition, and the distance to class centroids. Our attacks are also able to deal with integer constraints on the input, which naturally arise in domains like natural language. For example, our attacks on a linear support vector machine increase test error on the Enron spam detection dataset from 3% to 24% and on the IMDB sentiment classification dataset from 12% to 29% by adding just 3% poisoned data, even in the presence of these data sanitization defenses.

How do our attacks evade data sanitization defenses? Roughly speaking, the defenses we consider fall into two groups, which we handle separately. The first group of defenses uses anomaly detectors that are highly sensitive to the presence of just a few points: for instance, an anomaly detector that throws out points that are far away from their nearest neighbors will not recognize a particular point as anomalous if it is surrounded by a few other points, even if that small group of points is far from the rest of the data. Intuitively, such anomaly detectors tend to 'overfit' the training data. The second group of defenses uses anomaly detectors that are less sensitive; these detectors tend to be highly parametric,

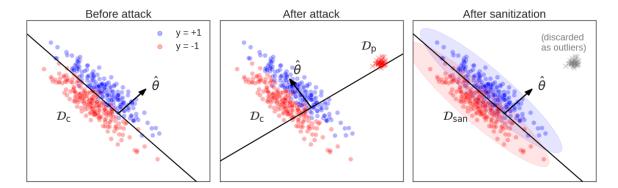


Figure 1: Left: In the absence of any poisoned data, the defender can often learn model parameters $\hat{\theta}$ that fit the true data \mathcal{D}_c well. Here, we show the decision boundary learned by a linear support vector machine on synthetic data. Middle: However, the addition of poisoned data \mathcal{D}_p can significantly change the learned $\hat{\theta}$, leading to high test error $\mathcal{L}(\hat{\theta})$. Right: By discarding outliers from $\mathcal{D} = \mathcal{D}_c \cup \mathcal{D}_p$ and then training on the remaining \mathcal{D}_{san} , the defender can mitigate the effectiveness of the attacker. In this example, the defender discards all blue points outside the blue ellipse, and all red points outside the red ellipse.

and tend to 'underfit' the training data. For example, consider an anomaly detector that throws away all points beyond a certain distance from the centroid of the data; whether this detector considers a given point as anomalous or not does not depend too much on the addition or removal of a few points from the data, as long as the data centroid does not change significantly.

To evade the first group of defenses, our attacks *concentrate* poisoned points in just a few distinct locations. These attacks tend to look normal to anomaly detectors that are more sensitive. For example, poisoned data that is placed in a tight cluster will evade the nearest-neighbor-based anomaly detector that throws out points far away from other points.

The second group of defenses is more resistant to concentrated attacks, since these defenses are less sensitive to small changes in the data (in this paper, we consider only attacks that inject 3% or less poisoned data, since in reality attackers might only have control over a small fraction of the training data). To evade this group of defenses, we formulate the data poisoning attack as a constrained optimization problem, where the attacker's objective is to maximize the test loss of the model that the defender learns on the union of the clean and poisoned data; the optimization variables are the locations of the poisoned points; and the constraints are imposed by the defenses (such that a point that satisfies the constraints will be guaranteed to evade the defenses).

Unfortunately for the attacker, this optimization problem is intractable to solve exactly (Bard, 1991), and even local methods like gradient ascent are slow (Biggio et al., 2012; Mei and Zhu, 2015b; Koh and Liang, 2017). To overcome this computational hurdle, we introduce two ideas:

1. We show theoretically that we can concentrate all of the attack mass on just a few distinct points (e.g., only 2 points for 2-class support vector machines (SVMs) and

logistic regression models) and still obtain an attack that is as effective as more diffuse attacks. This is more efficient as we have fewer optimization variables: we only need to optimize over the location of a few poisoned points instead of optimizing over all the points. This leads to what we call the **influence** attack (Section 4).

2. The difficulty in solving the above optimization problem is that the effect of the optimization variables (poisoned points) on the objective (test loss) is mediated by the model parameters that the defender learns on the poisoned data, an intermediate quantity that is expensive to compute. Instead, we propose a method for efficiently finding decoy parameters—model parameters that have high test error but low training error—and show that given fixed decoy parameters, we can efficiently find poisoned points such that training on these poisoned points yield those decoy parameters. We call this the **KKT** attack (Section 5), after the Karush-Kuhn-Tucker optimality conditions that we use to derive this attack. These decoy parameters also allow us to improve upon the attack introduced by Steinhardt et al. (2017), which leads to our **min-max** attack (Section 6).

Finally, our study reveals several surprising facts about data poisoning:

- 1. Poisoned data does not have to look anomalous; if the poisoned points are carefully coordinated, each poisoned point can appear normal, as in the above example of the nearest-neighbor based anomaly detector.
- Poisoned points need not have high loss under the poisoned model, and so the defender cannot simply throw out data points that have high loss. For example, given fixed decoy parameters, we can constrain our poisoned data to have low loss under the decoy parameters.
- 3. Regularization reduces the effect that any single data point can have on the model, and is therefore tempting to use as a defense against data poisoning. However, increasing regularization can actually make the defender more susceptible to attacks, because the defender becomes less able to fit the small fraction of poisoned points.

The success of our data poisoning attacks against common anomaly-based data sanitization defenses suggest that more work needs to be done on defending against data poisoning attacks. In particular, while anomaly detectors are typically well-suited to detect independently-generated anomalous points (e.g., due to some noise process in nature), a robust defense against data poisoning attacks will have to account for the ability of the attacker to place all of their poisoned data points in a coordinated fashion.

Beyond the merits of our specific attacks, we believe that our results underscore a broader and more important point: data poisoning attacks need to account for defenses, and defenses against such attacks correspondingly need to account for attacks that are specifically targeted against them. Attacks that do not consider defenses might work against a naive defender but be easily defeated by basic defenses. Similarly, defenses that are only tested against basic attacks might give a false sense of security, as they might be broken by more determined and coordinated attackers.

2. Problem Setting and Defenses

2.1 General setting

We consider classification tasks in this paper. For simplicity, we focus on binary tasks, though the ideas here generalize to multi-class tasks. In binary classification, the goal is to learn a mapping $f_{\theta}: \mathcal{X} \to \{-1, +1\}$, parametrized by θ , that maps from features $x \in \mathcal{X}$ to an output $y \in \{-1, +1\}$. We further assume that f_{θ} is a linear classifier, i.e., $f_{\theta}(x) = \text{sign}(\theta^{\top}x)$. A mapping f_{θ} is evaluated by its 0-1 test error $L_{0-1}(\theta; \mathcal{D}_{\text{test}})$ on some fixed test set $\mathcal{D}_{\text{test}} = \{(x_i, y_i)\}_{i=1}^{n_{\text{test}}}$, which is the proportion of points in $\mathcal{D}_{\text{test}}$ that it classifies wrongly:

$$L_{0-1}(\theta; \mathcal{D}_{\text{test}}) = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(x,y) \in \mathcal{D}_{\text{test}}} \mathbf{I}[f_{\theta}(x) \neq y]. \tag{1}$$

We model data poisoning as a zero-sum game between a defender, who wants to pick a $\hat{\theta}$ with low test error $L_{0\text{-}1}(\hat{\theta}; \mathcal{D}_{\text{test}})$, and an attacker, which wants to mislead the defender into picking a $\hat{\theta}$ with high $L_{0\text{-}1}(\hat{\theta}; \mathcal{D}_{\text{test}})$. The attacker observes the test set $\mathcal{D}_{\text{test}}$ as well as a clean training set $\mathcal{D}_{c} = \{(x_{i}, y_{i})\}_{i=1}^{n}$, and chooses ϵn poisoned points \mathcal{D}_{p} to add to \mathcal{D}_{c} . The defender observes the combined training set $\mathcal{D} = \mathcal{D}_{c} \cup \mathcal{D}_{p}$ consisting of the original n clean points and the ϵn additional poisoned points; uses a data sanitization defense to remove anomalous points; and then learns $\hat{\theta}$ from the remaining data.

Attacker:

- Input: Clean training data \mathcal{D}_{c} and test data \mathcal{D}_{test} .
- Output: Poisoned training data \mathcal{D}_p , with $|\mathcal{D}_p| = \epsilon |\mathcal{D}_c|$.
- Goal: Mislead defender into learning parameters $\hat{\theta}$ with high test error $L_{0-1}(\hat{\theta}; \mathcal{D}_{\text{test}})$.

Defender:

- Input: Combined training data $\mathcal{D} = \mathcal{D}_c \cup \mathcal{D}_p$.
- Output: Model parameters $\hat{\theta}$.
- Goal: Learn model parameters $\hat{\theta}$ with low test error $L_{0-1}(\hat{\theta}; \mathcal{D}_{test})$ by filtering out poisoned points \mathcal{D}_p .

In our setting, the attacker has several advantages: it knows the test set in advance (whereas the defender does not); it knows the defender's training procedure; and it also gets to observe the clean training set \mathcal{D}_c . In reality, the attacker might not have access to all of this information. However, as defenders, we want to be robust even to attackers that might have the above information (this is also known as the principle of security by design; see, e.g., Biggio et al. (2014)). For example, an attacker whose goal is to make the defender get a particular set of predictions wrong (e.g., the attacker might want to cause a "fake news" classifier to classify all websites from a certain domain as "real news") would accordingly

choose, and therefore get to observe, \mathcal{D}_{test} . In contrast, the defender might not know the attacker's goal in advance, and therefore would not have access to \mathcal{D}_{test} .

Our experiments in this paper consider the case where the test data \mathcal{D}_{test} is drawn from the same distribution as \mathcal{D}_c . This is known as an *indiscriminate* attack (Barreno et al., 2010), wherein the attacker tries to increase the average test error of the defender's model under the normal data distribution. Most of our methods are applicable to more general choices of \mathcal{D}_{test} ; we discuss this further in Section 8.

2.2 Data sanitization defenses

To thwart the attacker, we assume the defender employs a data sanitization defense (Cretu et al., 2008), which first removes anomalous-looking points from $\mathcal{D} = \mathcal{D}_c \cup \mathcal{D}_p$ and then trains on the remaining points. Why might a defender want to remove anomalies from the data before training? As a defender thinking from the attacker's perspective, the intuition is that poisoned data that looks similar to the clean data will not be effective in changing the learned model; therefore, the attacker would want to place poisoned points that are somehow different from the clean data. By discarding points that look too different, the defender can therefore protect itself against attack.

Different defenses differ in how they judge points as being anomalous: for example, in what we call the **L2** defense, the defender first finds the centroids of each class in \mathcal{D} , and then discards points in \mathcal{D} that are far from their respective class centroids. To formalize this, we represent each defense by a score function $s_{\beta}: \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ that takes in a data point (x,y) and returns a number representing how anomalous that data point is. This score function is parametrized by anomaly detector parameters β that are derived from the combination of the clean and poisoned training data $\mathcal{D} = \mathcal{D}_c \cup \mathcal{D}_p$. In the **L2** defense, we would have $\beta = (\mu_+, \mu_-)$ represent the class centroids and $s_{\beta}(x,y) = ||x - \beta_y||_2$ measure the distance of x to the centroid of class y.

Concretely, the defender:

- 1. Fits the anomaly detector parameters $\beta = B(\mathcal{D})$, where B is a function (specific to a particular defense) that takes in a dataset and returns a vector.
- 2. Constructs the feasible set $\mathcal{F}_{\beta} = \{(x,y) : (x,y) \in \mathcal{X} \times \mathcal{Y} \text{ with } s_{\beta}(x,y) < \tau_y\}$. We assume that the threshold τ_y is chosen such that a desired fraction of points from each class are discarded (e.g., the defender might choose to discard the most 5% of the training points from each class).
- 3. Forms the sanitized training dataset $\mathcal{D}_{san} = \mathcal{D} \cap \mathcal{F}_{\beta}$ by discarding all points that fall outside the feasible set.
- 4. Finds the $\hat{\theta}$ that minimizes the regularized training loss on \mathcal{D}_{san} :

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} L(\theta; \mathcal{D}_{\operatorname{san}}) \stackrel{\text{def}}{=} \underset{\theta}{\operatorname{argmin}} \frac{\lambda}{2} \|\theta\|_{2}^{2} + \frac{1}{|\mathcal{D}_{\operatorname{san}}|} \sum_{(x,y) \in \mathcal{D}_{\operatorname{san}}} \ell(\theta; x, y), \tag{2}$$

where λ is a hyperparameter controlling regularization strength and ℓ is a convex surrogate for the 0/1-loss that f_{θ} incurs. In this paper, we mainly consider support vector machines (SVMs), which use the hinge loss $\ell(\theta; x, y) = \max(0, 1 - y\theta^{\top}x)$.

Ideally—as in the hypothetical scenario in Figure 1—a defense would filter out the poisoned data \mathcal{D}_p and leave \mathcal{D}_{san} close to the clean data \mathcal{D}_c , so that the defender learns model parameters $\hat{\theta}$ that have low test error.

As stated in Section 2.1, the attacker's goal is to pick poisoned points that mislead the defender into choosing model parameters that incur high test error. Against a defender that employs the data sanitization defense described above, we can formulate the attacker's goal as the following optimization problem:

maximize
$$L_{0-1}(\hat{\theta}; \mathcal{D}_{test})$$
 (3)
s.t. $|\mathcal{D}_{p}| = \epsilon |\mathcal{D}_{c}|$, (ϵ fraction of poisoned points)
where $\beta = B(\mathcal{D}_{c} \cup \mathcal{D}_{p})$ (\mathcal{F}_{β} is fit on clean and poisoned data)
 $\hat{\theta} \stackrel{\text{def}}{=} \underset{\beta}{\operatorname{argmin}} L(\theta; (\mathcal{D}_{c} \cup \mathcal{D}_{p}) \cap \mathcal{F}_{\beta})$. (defender trains on remaining data)

The first constraint corresponds to the attacker only being able to add in an ϵ fraction of poisoned points; the second constraint corresponds to the defender fitting the anomaly detector on the entire training set $\mathcal{D}_c \cup \mathcal{D}_p$; and the final equality corresponds to the defender learning model parameters $\hat{\theta}$ that minimize training loss.

In this work, we consider 5 different data sanitization defenses, chosen to span a broad range of approaches to data sanitization and anomaly detection. Each defense is parametrized by different score functions s_{β} and anomaly detector parameters $\beta = B(\mathcal{D})$:

• The L2 defense removes points that are far from their class centroids in L_2 distance:

$$\beta_y = \mathbb{E}_{\mathcal{D}}[x|y]$$

$$s_{\beta}(x,y) = \|x - \beta_y\|_2$$

• The slab defense (Steinhardt et al., 2017) first projects points onto the line between the class centroids, and then removes points that are too far from the class centroids:

$$\beta_y = \mathbb{E}_{\mathcal{D}}[x|y]$$

$$s_{\beta}(x,y) = \left| (\beta_1 - \beta_{-1})^{\top} (x - \beta_y) \right|$$

The idea behind this defense is that we only want to look at the relevant dimensions in feature space to find outliers. The **L2** defense treats all dimensions equally, whereas the **slab** defense treats the vector between the class centroids as the only relevant dimension.

• The **loss** defense discards points that are not well fit by a model trained (without any data sanitization) on the full dataset \mathcal{D} :

$$\beta = \operatorname*{argmin}_{\theta} \mathbb{E}_{\mathcal{D}}[\ell_{\theta}(x, y)]$$

$$s_{\beta}(x, y) = \ell_{\beta}(x, y)$$

For a linear model, this is conceptually similar to the **slab** defense, except that the relevant feature dimension is learned using the loss function instead of being fixed as the direction between the class centroids.

• The **SVD** defense assumes that the clean data lies in some low-rank subspace, and that poisoned data therefore will have a large component out of this subspace (Rubinstein et al., 2009). Let X be the data matrix, with the i-th row containing x_i , the features of the i-th training point. Then:

$$\beta = \text{Matrix of top } k \text{ right singular vectors of } X$$

$$s_{\beta}(x, y) = \|(I - \beta \beta^{\top})x\|_{2}$$

In our experiments, we choose the smallest k such that the normalized Frobenius approximation error (i.e., the normalized sum of the squared singular values) is 0.05.

• The k-NN defense removes points that are far from their k nearest neighbors.

$$eta = \mathcal{D}_{\mathrm{c}} \cup \mathcal{D}_{\mathrm{p}}$$
 $s_{\beta}(x,y) = \text{Distance to k-th nearest neighbor in β}$

In our experiments, we set k = 5.

Note that β is sometimes a simple set of summary statistics of the dataset (e.g., in the **L2** and **slab** defenses), while at other times β can be the entire dataset (e.g., in the **k-NN** defense). We will handle these two types of defenses separately, as we discuss in Section 3.

Automated defenses. We are interested in settings where the volume of data is too large for humans to manually inspect. We therefore consider automated defense systems that rely on the above statistical rules for anomaly detection. Note that this means that a valid poisoned point, under our framework, could still look anomalous to a human expert (e.g., a poisoned point with a bag-of-words feature representation might not correspond to any grammatical sentence), though it will obey basic input constraints (e.g., having an integer number of words).

Attack evaluation. The attacker's goal is to increase test error regardless of which defense is deployed against it. To evaluate an attack, we run each of the above defenses separately against it. For an attack \mathcal{D}_p to be considered successful, it needs to significantly increase test error against all of the defenses.

Attack budget and defense threshold. We assume that the attacker has limited control over the training data: in our experiments, we allow the attacker to only add up to $\epsilon = 3\%$ poisoned data, and we set τ to remove p = 5% of the training data from each class.

2.3 Datasets and input constraints

In our experiments, we use 4 datasets for our binary classification tests: the MNIST-1-7 (LeCun et al., 1998) and Dogfish (Koh and Liang, 2017) image datasets, and the Enron spam detection (Metsis et al., 2006) and IMDB sentiment classification datasets (Maas et al., 2011). These are the 4 datasets considered in Steinhardt et al. (2017), which also studied data poisoning. Summaries of each dataset are given in Table 1, including the number of training points n, the dimension of each point d, and the base accuracy of an SVM trained only on the clean data. Each dataset has its own characteristics and input constraints:

Dataset	Classes	$\mid n \mid$	n_{test}	d	Base Error	Input Constraints
MNIST-1-7	2	13007	2163	784	$0.7\% \ (\lambda = 0.01)$	$\boxed{[0,1]}$
Dogfish	2	1800	600	2048	$1.3\% \ (\lambda = 1.10)$	\mathbf{R}
Enron	2	4137	1035	5116	$2.9\% \ (\lambda = 0.09)$	$\mathbf{Z}_{\geq 0}$
IMDB	2	25000	25000	89527	$11.9\% \ (\lambda = 0.01)$	$\mathbf{Z}_{>0}^-$
MNIST	10	60000	10000	784	$7.5\%^{1}$	[0,1]

Table 1: Characteristics of the datasets we consider, together with the base test errors that an SVM achieves on them (with regularization parameters λ selected by validation). The input covariates for Enron and IMDB must be non-negative integers.

- The MNIST-1-7 image dataset (LeCun et al., 1998) requires each input feature to lie within the interval [0, 1] (representing normalized pixels). It is derived from the standard 10-class MNIST dataset by taking just the images labeled '1' or '7'. It is easily linearly separable: an SVM achieves 0.7% error on the clean data.
- The Dogfish image dataset (Koh and Liang, 2017) has no input constraints; its features are neural network representations, which for the purposes of this paper we allow to take any value in \mathbb{R}^d . Compared to the MNIST-1-7 dataset (where $n \gg d$), the Dogfish dataset has $n \approx d$, allowing attackers to potentially exploit overfitting. The Dogfish dataset also has a low base error of 1.3%.
- The Enron spam classification text dataset (Metsis et al., 2006) requires input to be non-negative and integer valued, as each feature represents word frequency. As with the Dogfish dataset, the Enron dataset has $n \approx d$ and a relatively low base error of 3.0%.
- The IMDB sentiment classification text dataset (Maas et al., 2011) similarly requires input to be non-negative and integer valued. Compared to the other datasets, the IMDB dataset has significantly larger n and d, presenting computational challenges. It also has $n \ll d$ and is not as linearly separable as the other datasets, with a high base error of 11.9%.

We also use the standard 10-class MNIST dataset (LeCun et al., 1998) for multi-class experiments, which we discuss in Section 6.

Input constraints. The feasible set \mathcal{F}_{β} encodes both the defenses and the input constraints of the dataset, since $\mathcal{F}_{\beta} \subseteq \mathcal{X} \times \mathcal{Y}$ and the input domain \mathcal{X} only includes valid points. Thus, the defender will eliminate all input points that do not obey the input constraints of the dataset.

^{1.} We use the multi-class SVM formulation in Crammer and Singer (2002) to classify the 10-class MNIST dataset, with no explicit regularization. For computational reasons, we use AdaGrad (Duchi et al., 2010) to optimize the parameters for the multi-class problem, which provides implicit regularization.

3. Attack Framework

In this paper, we take on the role of the attacker. Recall that we are given a set of n clean training points \mathcal{D}_c and a test set \mathcal{D}_{test} , and our goal is to come up with a set of ϵn poisoned training points \mathcal{D}_p such that a defender following the procedure in Section 2.2 will choose model parameters $\hat{\theta}$ that incur high test error $\mathcal{L}(\hat{\theta})$. The difficulty lies in choosing poisoned points \mathcal{D}_p that will both lead to high test error and also avoid being flagged as anomalous.

As mentioned in Section 1, we can roughly group anomaly detectors into two categories: those that are more sensitive to the data and tend to 'overfit' (like the **k-NN** defense), and those that are less sensitive to the data and tend to 'underfit' because they make strong parametric assumptions (like the **L2** defense). This categorization is neither precise nor rigorous, but it provides useful intuition. As we discuss in this section, we can evade defenses in the first category by using concentrated attacks, and we can evade defenses in the second category by using constrained optimization.

Outline. In this section, we will introduce the general ideas that underlie each of the three data poisoning attacks that we develop in this paper. Specifically, we discuss concentrated attacks in Section 3.1; formulating data poisoning as a constrained optimization problem in Section 3.2; and handling integer input constraints in Section 3.3. Subsequently, in Sections 4 to 6, we will detail three specific methods—corresponding to the three different attacks—for efficiently solving the optimization problem described in this section.

3.1 Concentrated attacks

To bypass anomaly detectors that are sensitive to small changes in the data, we observe that poisoned data that is *concentrated* on a few locations tends to appear normal. This is a simple observation, but to be the best of our knowledge has yet to be exploited for data poisoning. For example:

- For the **k-NN** defense and other similar nonparametric anomaly detectors, this is trivially true: if several poisoned points are placed very near each other, then by definition, the distances to their nearest neighbors will be small.
- For the **SVD** defense, it is more likely that the low-rank representation of \mathcal{D} will include the poisoned points, reducing their out-of-projection components.
- For the **loss** defense, if the poisoned points are concentrated in a similar location, the model will have more incentive to fit those points (because fitting one of them would imply fitting all of them, which would reduce the training loss more than fitting a single isolated point).

In general, anomaly detectors flag points that do not look like other points in the training data, and can therefore be fooled by sufficiently concentrated groups of poisoned points (because each poisoned point in each group lies close to other poisoned points). For example, an attacker could evade the \mathbf{k} - $\mathbf{N}\mathbf{N}$ defense by placing tightly-clustered groups of poisoned points of size at least k.

However, from the attacker's point of view, one potential issue is that being constrained to place points in concentrated groups might hurt the attacker's ability to change the learned

model, as measured by the number of poisoned points required to make the defender learn some target parameters. For example, it might be the case that an efficient attack would involve spreading out each poisoned point throughout the feasible set; and thus if the attacker were to be forced to group these points together, the resulting attack might be less effective at changing the defender's model.

Our theoretical contribution is to show that if the feasible sets for each class are convex, and if the defender is using a 2-class SVMs or logistic regression model, then—fortunately for the attacker—the above scenario will not occur. In particular, in that setting, we would only need two distinct points (one per class) to realize any attack:

Theorem 1 (2 points suffice for 2-class SVMs and logistic regression) Consider a defender that learns a 2-class SVM or logistic regression model by first discarding all points outside a fixed feasible set \mathcal{F} , and then finding the parameters that minimize the average (regularized) training loss. Suppose that for each class y = -1, +1, the feasible set $\mathcal{F}_y \stackrel{\text{def}}{=} \{x : (x,y) \in \mathcal{F}\}$ is a convex set. If a parameter $\hat{\theta}$ is attainable by any set of \tilde{n} poisoned points $\mathcal{D}_p = \{(\tilde{x}_1, \tilde{y}_1), \dots, (\tilde{x}_{\tilde{n}}, \tilde{y}_{\tilde{n}})\} \subseteq \mathcal{F}$, then there exists a set of at most \tilde{n} poisoned points $\tilde{\mathcal{D}}_p$ (possibly with fractional copies) that also attains $\hat{\theta}$ but only contains 2 distinct points, one from each class.

More generally, the above statement is true for any margin-based model with loss of the form $\ell(\theta; x, y) = c(-y\theta^{\top}x)$, where $c : \mathbb{R} \to \mathbb{R}$ is a convex, monotone increasing, and twice-differentiable function, and the ratio of second to first derivatives c''/c' is monotone non-increasing.

Remark 2 If we, as the attacker, concentrate all of the poisoned points in only two distinct locations, we will trivially evade the k-NN defense. The remaining defenses—L2, slab, loss, and SVD—all have convex feasible sets, so the conditions of the theorem hold. One technicality is that in reality, the feasible set will not remain fixed as the theorem assumes: if an attack \mathcal{D}_p is collapsed into a different attack $\tilde{\mathcal{D}}_p$ with at most 2 distinct points, the feasible set \mathcal{F}_{β} will also change. However, as discussed at the beginning of this section, it will tend to change in a way that makes the poisoned points feasible, so if the original attack \mathcal{D}_p was already feasible, the new attack $\tilde{\mathcal{D}}_p$ will likely also be feasible.

We defer the full proof to Appendix A. As a short proof sketch, the proof consists of two parts: we first relate the number of distinct points necessary to achieve an attack to the geometry of the set of feasible gradients (i.e., the set of gradients of points within the feasible set), using the notion of Carathéodory numbers from convex geometry. We then show, for the specific losses considered above (the hinge and logistic loss), that this set of feasible gradients has the necessary geometry. Our method is general and can be extended to different loss functions and feasible sets; we provide one such extension, to a multi-class SVM setting, in Appendix A.

The idea that concentrated points can evade anomaly detectors is separate from the idea that only two distinct points are needed to realize any given attack. To summarize:

1. Concentrated attacks tend to evade anomaly detectors that are more sensitive to changes in the data, including the **k-NN**, **SVD**, and **loss** defenses.

2. By Theorem 1, concentrating an attack does not diminish its efficiency (as measured by the number of poisoned points needed to make the defender achieve some target parameters $\hat{\theta}$).

In Section 4, we show empirically that such concentrated attacks can indeed be effective at poisoning a defender's model while evading the **k-NN**, **SVD**, and **loss** defenses.

One objection to attacks with only two distinct poisoned points is that they can be easily defeated by a defender that throws out repeated points. However, the attacker can add a small amount of random noise to fuzz up poisoned points without sacrificing the concentrated nature of the attack. Randomized rounding, which we use to handle datasets with integer input constraints in Section 3.3, is a version of this procedure.

3.2 Constrained optimization

Anomaly detectors that are more robust to small changes in the training data, such as those that rely on simple sufficient statistics of the data, tend to be less vulnerable to concentrated attacks. In our setting, the **L2** and **slab** defenses in particular are not fooled by concentrated attacks: the class centroid, which is used in both defenses, cannot be moved too much by an ϵ fraction of poisoned points if ϵ is small and the clean training data \mathcal{D}_c well-clustered. (The class centroid cannot be arbitrarily changed by a single poisoned point that is sufficiently far away, as such a point would be filtered out by the **L2** defense.)

To handle the **L2** and **slab** defenses, we formulate the attacker's goal as an iterative constrained optimization problem. At a high level, at each iteration, we update the feasible set \mathcal{F}_{β} based on the current poisoned data \mathcal{D}_{p} , and then optimize for a new set of poisoned data \mathcal{D}'_{p} within this feasible set that maximizes test loss while still evading detection. Because these two defenses are relatively stable to small changes in the data, the feasible set \mathcal{F}_{β} does not change too much from iteration to iteration, which makes this iterative optimization feasible.

We start with the optimization problem (3) from Section 2.2, which we reproduce here for convenience:

$$\begin{aligned} & \underset{\mathcal{D}_{p}}{\text{maximize}} & & L_{0\text{-}1}(\hat{\theta}; \mathcal{D}_{test}) \\ & \text{s.t.} & & |\mathcal{D}_{p}| = \epsilon |\mathcal{D}_{c}| \\ & & \beta = B(\mathcal{D}_{c} \cup \mathcal{D}_{p}), \\ & \text{where} & & \hat{\theta} \overset{\text{def}}{=} \underset{\theta}{\text{argmin}} L\left(\theta; (\mathcal{D}_{c} \cup \mathcal{D}_{p}) \cap \mathcal{F}_{\beta}\right). \end{aligned}$$

To make this problem more tractable, we make two approximations. First, we assume that the defender does not discard any clean points. Thus, if all poisoned points are constrained to lie within the feasible set \mathcal{F}_{β} and therefore evade sanitization, then the defender trains on the entirety of $\mathcal{D} = \mathcal{D}_c \cup \mathcal{D}_p$. Second, we replace the 0-1 test error $L_{0-1}(\hat{\theta}; \mathcal{D}_{test})$ with its convex surrogate $L(\hat{\theta}; \mathcal{D}_{test}) \stackrel{\text{def}}{=} \mathbb{E}_{\mathcal{D}_{test}} \ell(\hat{\theta}; x, y)$, which is continuous and easier to optimize.

^{2.} This favors the defender, since we do not consider the case in which a savvy attacker might place poisoned points in such a way as to cause the defender to throw out particularly good points in \mathcal{D}_c and therefore learn a bad model.

These approximations let us rewrite the above optimization problem as the following:

maximize
$$L(\hat{\theta}; \mathcal{D}_{test})$$
 (4)
s.t. $|\mathcal{D}_{p}| = \epsilon |\mathcal{D}_{c}|$
 $\mathcal{D}_{p} \subseteq \mathcal{F}_{\beta}$
 $\beta = B(\mathcal{D}_{c} \cup \mathcal{D}_{p}),$
where $\hat{\theta} \stackrel{\text{def}}{=} \underset{\theta}{\operatorname{argmin}} L(\theta; \mathcal{D}_{c} \cup \mathcal{D}_{p}).$

This problem is still difficult to solve directly; one issue is that the feasible set \mathcal{F}_{β} depends on the poisoned data \mathcal{D}_{p} . However, because the feasible sets of the **L2** and **slab** defenses depend on the class centroids, which are stable with respect to small changes in the poisoned data \mathcal{D}_{p} , we can alternate between optimizing over \mathcal{D}_{p} for a fixed β , and then updating β to reflect the new \mathcal{D}_{p} (Algorithm 1). This iterative optimization procedure is guaranteed to make progress so long as the poisoned points \mathcal{D}_{p} remain valid even after re-fitting β .

```
Algorithm 1 Pseudocode for attack via constrained optimization.
```

```
Input: clean data \mathcal{D}_{c}, poisoned fraction \epsilon.

Initialize anomaly detector on clean data: \beta \leftarrow B(\mathcal{D}_{c})

repeat

\mathcal{D}_{p} \leftarrow \underset{|\mathcal{D}_{p}|=\epsilon|\mathcal{D}_{c}|}{\operatorname{argmax}} L(\hat{\theta}; \mathcal{D}_{test}) \quad \text{(Solve (4) with fixed } \beta; \text{ see Sections 4-6)}
\beta \leftarrow B(\mathcal{D}_{p}) \quad \text{(Update anomaly detector } \beta \text{ with new } \mathcal{D}_{p})
until \mathcal{D}_{p} converges
Output \mathcal{D}_{p}.
```

The stability of the **L2** and **slab** defenses makes this problem easier to optimize, because it implies that β does not change too much between iterations. In practice, these defenses are so stable to small sets of poisoned data \mathcal{D}_p that we can even skip the iterative optimization. Instead, we fix the anomaly detector $\beta = B(\mathcal{D}_c)$ on the clean data and solve (4) once. This non-iterative formulation still yields effective attacks, though the iterative optimization does make the attacks slightly better (Section 4). This non-iterative constrained optimization formulation has been used in prior work (e.g., Steinhardt et al. (2017)); our main contribution in this regard is developing more effective and computationally efficient ways of solving the optimization problem (Sections 4 to 6).

3.3 Handling integer input constraints with randomized rounding

Each of the three data poisoning attacks that we will subsequently develop use some form of gradient descent on the poisoned points to solve (4) (the first step in Algorithm 1). However, gradient descent cannot be directly applied in settings where the input features are constrained to be non-negative integers (e.g., with bag-of-word models in natural language tasks). To handle this, we propose a linear programming (LP) relaxation specifically designed to keep the poisoned points from being detected.

Our general approach is similar to how Steinhardt et al. (2017) handle integer-valued input: we relax the integrality constraint to allow non-negative real-valued points, and then perform randomized rounding on these real-valued points to obtain integer-valued points. More precisely, we:

- 1. Solve the optimization problem (4) while allowing all poisoned points $(x, y) \in \mathcal{D}_p$ to have real-valued x.
- 2. Then, for each $(x, y) \in \mathcal{D}_p$, we construct a rounded \hat{x} as follows: for each coordinate i, $\hat{x}_i = \lceil x_i \rceil$ with probability $x_i \lfloor x_i \rfloor$, and $\hat{x}_i = \lfloor x_i \rfloor$ otherwise. ($\lceil \cdot \rceil$ denotes ceiling and $| \cdot |$ denotes floor.)

Note that this procedure preserves the mean: $\mathbb{E}[\hat{x}] = x$. However, randomized rounding can produce poisoned points that get detected by data sanitization defenses. We introduce two techniques to avoid this:

Repeated points. In high dimensions, randomized rounding can result in poisoned points that are far away from each other. This can make the poisoned points vulnerable to being filtered out by the defender. Instead, as we discuss in Section 3.1, we want the poisoned points to be concentrated in a few distinct locations. To do so, we adopt a heuristic of repeating poisoned points r times after rounding (keeping the overall fraction of poisoned data, ϵ , the same). This means that we find $\epsilon n/r$ poisoned points $\{x_1, x_2, \ldots, x_{\epsilon n/r}\}$, do randomized rounding to get $\{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{\epsilon n/r}\}$, and then form the multiset \mathcal{D}_p by taking the union of r copies of this set. In practice, we find that setting r = 2 or 3 works well. This heuristic better concentrates the poisoned points while still preserving their expected mean $\mathbb{E}[\hat{x}]$.

Linear programming (LP) relaxation for the L2 defense. Randomized rounding violates the L_2 constraint used in the L2 defense. Recall that in the L2 defense, we wish to play points (x, y) such that $||x - \mu_y||_2 \le \tau_y$, where μ_y is the mean of class y and τ_y is some threshold (Section 2.2). The issue is that even if we control the norm of the continuous x by having $||x - \mu_y||_2 \le \tau_y$, we could still have the randomly-rounded \hat{x} violate this constraint on expectation: $\mathbb{E}[||\hat{x} - \mu_y||_2] > \tau_y$. This is because the L_2 norm is convex, so by Jensen's inequality, $\mathbb{E}[||\hat{x} - \mu_y||_2] \ge ||\mathbb{E}[\hat{x}] - \mu_y||_2 = ||x - \mu_y||_2$.

As a result, the rounded \hat{x} could still have a high chance of being detected by the **L2** defense, even if the continuous x is safe. Steinhardt et al. (2017) deal with this problem by setting τ_y conservatively, so that \hat{x} might avoid detection even if $\|\hat{x} - \mu_y\|_2 > \|x - \mu_y\|_2$. However, the conservative threshold reduces the attacker's options, resulting in a generally less effective attack.

To handle the effect of rounding in a principled way, it suffices to control the expected squared norm $\mathbb{E}[\|\hat{x}-\mu_y\|_2^2]$: if $\mathbb{E}[\|\hat{x}-\mu_y\|_2^2] < \tau_y^2$, then by Jensen's inequality, $\mathbb{E}[\|\hat{x}-\mu_y\|_2] < \tau_y$. To compute the expected squared norm, we first write

$$\mathbb{E}[\|\hat{x} - \mu_y\|_2^2] = \mathbb{E}[\|\hat{x}\|_2^2] - 2\langle x, \mu_y \rangle + \|\mu_y\|_2^2.$$
 (5)

Note that $\mathbb{E}[\|\hat{x}\|_2^2]$ can in general be substantially larger than $\|x\|_2^2$ due to the variance from rounding. We can compute $\mathbb{E}[\|\hat{x}\|_2^2]$ explicitly as

$$\mathbb{E}[\|\hat{x}\|_{2}^{2}] = \sum_{i=1}^{d} x_{i}(\lceil x_{i} \rceil + \lfloor x_{i} \rfloor) - \lceil x_{i} \rceil \lfloor x_{i} \rfloor. \tag{6}$$

While the function $f(x) \stackrel{\text{def}}{=} x(\lceil x \rceil + \lfloor x \rfloor) - \lceil x \rceil \lfloor x \rfloor$ looks complicated, it actually has a nice form, which we can see by plotting it:

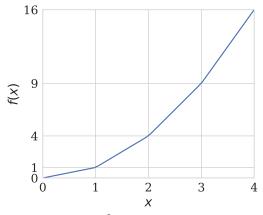


Figure 2: Plot of $\mathbb{E}[\|\hat{x}\|_2^2] = f(x)$ against x for scalar x.

Intuitively, we expect f(x) to be close to x^2 . Indeed, as Figure 2 shows, it is a piecewise-linear function where the k-th piece linearly interpolates between k^2 and $(k+1)^2$, and we can write it as the maximum over a set of linear equations:

$$f(x) = \max_{k=0}^{\infty} (2k+1)x - k(k+1). \tag{7}$$

Thus, when solving the attacker optimization (equation (4)) for datasets that have non-negative integer constraints, we replace the standard **L2** feasible set $\mathcal{F}_{\beta} = \{(x,y) : \|x - \mu_y\|_2 \le \tau_2 \text{ and } x \in \mathbf{R}_{\ge 0}\}$ with the modified constraint set

$$\mathcal{F}_{LP} = \{ (x, y) : \mathbb{E}[\|\hat{x} - \mu_y\|_2^2] \le \tau_y^2 \text{ and } x \in \mathbf{R}_{\ge 0} \}.$$
 (8)

If we approximate the infinite maximum in (7) by its first K terms, then the corresponding approximation of \mathcal{F}_{LP} can be represented via a linear program. In our experiments, we choose K adaptively for each coordinate i to be equal to the largest value that x_i attains across the dataset. This formulation allows us to express the L_2 norm constraint $\mathbb{E}[\|\hat{x} - \mu_y\|_2^2] \leq \tau_y^2$ as a set of linear constraints on the continuous x, allowing us to control the expected L_2 norm of the poisoned points after rounding.

Randomized rounding has some negative impact: it makes attacks less concentrated, as discussed above, and can also result in a few unlucky poisoned points getting filtered by other defenses (e.g., by the **loss** defense if the rounding happens to increase the loss on the point). Another advantage of the above LP relaxation is that in practice, the linear constraints tend to lead to nearly-integer x, which further reduces the negative impact of having to do randomized rounding.

4. Improved Influence Attack

In the remainder of this paper, we discuss the task of solving the optimization problem (4) with a fixed β (the first step in Algorithm 1). This problem is a bilevel optimization problem (so named because the outer maximization involves solving an inner minimization); such problems are, in general, non-convex and intractable to solve exactly (Bard, 1991, 1999). In this section and in Sections 5 and 6, we discuss three different methods for efficiently approximating this problem, corresponding to the three different attacks that we develop. These methods all generate concentrated attacks (Section 3.1) within our constrained optimization framework (Section 3.2), and are therefore able to evade all five of the defenses that we consider (Section 2.2).

Recall that the optimization problem (4) involves finding poisoned data \mathcal{D}_p that maximizes the defender's test loss $L(\hat{\theta}; \mathcal{D}_{test})$ for a fixed feasible set \mathcal{F}_{β} (Section 3.2). In this section, we introduce the **influence** attack, which solves this task through projected gradient ascent.

At a high level, we can find a local maximum of (4) by iteratively taking gradient steps on the features of each poisoned point in \mathcal{D}_p , projecting each point onto the feasible set \mathcal{F}_{β} after each iteration. This type of gradient-based data poisoning attack has been previously studied in the literature (see, e.g., Biggio et al. (2012) and Mei and Zhu (2015b) for attacks without the projection step, and Koh and Liang (2017) and Steinhardt et al. (2017) for attacks with the projection step). Koh and Liang (2017) use influence functions to compute this gradient; accordingly, we call this projected gradient ascent method the influence attack. Our method builds upon previous work by incorporating the techniques mentioned in Section 3—concentrating the attack and using randomized rounding with the LP relaxation.

4.1 The basic influence attack

First, we review the basic influence attack, borrowing from the presentation in Koh and Liang (2017). Our goal is to perform gradient ascent on each $(\tilde{x}, \tilde{y}) \in \mathcal{D}_p$ to maximize the test loss $L(\hat{\theta}; \mathcal{D}_{test})$. To do so, we need to compute the gradient of $L(\hat{\theta}; \mathcal{D}_{test})$ w.r.t. each (\tilde{x}, \tilde{y}) . Since the labels \tilde{y} are discrete, we cannot compute gradients on them; we therefore fix the labels \tilde{y} at the beginning of the algorithm, and only optimize over the covariates \tilde{x} .

Computing the gradient. The difficulty in computing the gradient of the test loss $L(\hat{\theta}; \mathcal{D}_{\text{test}})$ w.r.t. each \tilde{x} in \mathcal{D}_{p} is that L depends on \tilde{x} only through the model parameters $\hat{\theta}$, which is a complicated function of \mathcal{D}_{p} . To get around this, the influence attack uses a closed-form estimate of $\frac{\partial \hat{\theta}}{\partial \tilde{x}}$, which measures how much the model parameters $\hat{\theta}$ change with a small change to \tilde{x} . The desired derivative $\frac{\partial L}{\partial \tilde{x}}$ can then be computed via the chain rule: $\frac{\partial L}{\partial \tilde{x}} = \frac{\partial L}{\partial \hat{\theta}} \frac{\partial \hat{\theta}}{\partial \tilde{x}}$.

The quantity $\frac{\partial L}{\partial \hat{\theta}}$ is the average gradient of the loss over the test set, which we denote as $g_{\hat{\theta}, \mathcal{D}_{\text{test}}}$ for convenience, and it can be computed straightforwardly as

$$\frac{\partial L}{\partial \hat{\theta}} = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(x,y) \in \mathcal{D}_{\text{test}}} \nabla \ell(\hat{\theta}; x, y) \stackrel{\text{def}}{=} g_{\hat{\theta}, \mathcal{D}_{\text{test}}}.$$
 (9)

 $\frac{\partial \hat{\theta}}{\partial \tilde{x}}$ is more involved. As shown in Section 2.2 of Koh and Liang (2017), this latter derivative can be computed as

$$\frac{\partial \hat{\theta}}{\partial \tilde{x}} = -H_{\hat{\theta}}^{-1} \frac{\partial^2 \ell(\hat{\theta}; \tilde{x}, \tilde{y})}{\partial \hat{\theta} \partial \tilde{x}},\tag{10}$$

where $H_{\hat{\theta}}$ is the Hessian of the training loss at $\hat{\theta}$:

$$H_{\hat{\theta}} \stackrel{\text{def}}{=} \lambda I + \frac{1}{|\mathcal{D}_{c} \cup \mathcal{D}_{p}|} \sum_{(x,y) \in \mathcal{D}_{c} \cup \mathcal{D}_{p}} \frac{\partial^{2} \ell(\hat{\theta}; x, y)}{\partial \hat{\theta}^{2}}. \tag{11}$$

Combining equations (9) to (11), the gradient of the test loss w.r.t. an attack point \tilde{x} is

$$\frac{\partial L(\hat{\theta})}{\partial \tilde{x}} = -g_{\hat{\theta}, \mathcal{D}_{\text{test}}}^{\top} H_{\hat{\theta}}^{-1} \frac{\partial^2 \ell(\hat{\theta}; \tilde{x}, \tilde{y})}{\partial \hat{\theta} \, \partial \tilde{x}}.$$

Projecting onto the feasible set. To maintain the feasibility of each poisoned point $(\tilde{x}, \tilde{y}) \in \mathcal{D}_p$, we project them onto the feasible set \mathcal{F}_{β} by setting it to the feasible point $(x, \tilde{y}) \in \mathcal{F}_{\beta}$ that is closest in ℓ_2 distance to (\tilde{x}, \tilde{y}) . In our setting, the feasible set is given by the **L2** and **slab** defenses, so \mathcal{F}_{β} is convex; thus, this projection is well-defined, in the sense that there is always a unique closest point in \mathcal{F}_{β} . Finding this projection is a convex optimization problem that can be solved efficiently by a general-purpose convex solver.

Algorithm. The full algorithm iteratively computes the gradient $\frac{\partial L}{\partial \tilde{x}}$ for each attack point $(\tilde{x}, \tilde{y}) \in \mathcal{D}_p$, moves in the direction of the gradient, and then projects back onto the feasible set \mathcal{F}_{β} . In Algorithm 2, we give pseudocode for interleaving this gradient computation and projection into the iterative optimization framework of Section 3.2 by updating the feasible set \mathcal{F}_{β} after each iteration. We call our specific implementation, described below, the **influence-basic** attack.

Implementation details. We initialize both \tilde{x}_i and \tilde{y}_i through random label flips on the training set, selecting ϵn data points from the clean data \mathcal{D}_c uniformly at random, with replacement, to flip and add to the poisoned data \mathcal{D}_p . We only consider data points that would still lie in \mathcal{F}_{β} after the label flip.

We take the loss $\ell(\theta;x,y)$ to be the hinge loss $\max(0,1-y\theta^{\top}x)$. To efficiently compute $\frac{\partial \hat{\theta}}{\partial \bar{x}}$ in (10), we follow Koh and Liang (2017) and use a combination of fast Hessian-vector products (Pearlmutter, 1994) and a conjugate gradient solver (Martens, 2010); see also Agarwal et al. (2016) and Muñoz-González et al. (2017) for other tractable approaches. We select the step size η by trying a range of options and selecting the best-performing one on the test set (since, in our setting, the attacker knows the test set in advance).

We cannot do gradient descent directly on datasets where the \tilde{x} 's are constrained to take on integer values, since the \tilde{x} 's are not continuous in that case. To handle this, the **influence-basic** attack uses the vanilla randomized rounding procedure described in Section 3.3 (without the repeated points heuristic or linear programming relaxation).

4.2 Improvements to the basic algorithm

The **influence-basic** attack suffers from several drawbacks. To address these, we introduce two new improvements to the algorithm.

Algorithm 2 The influence-basic attack.

```
Input: clean data set \mathcal{D}_c, poisoning fraction \epsilon, step size \eta.

Initialize poisoned data set \mathcal{D}_p \leftarrow \{(\tilde{x}_1, \tilde{y}_1), \dots, (\tilde{x}_{\epsilon n}, \tilde{y}_{\epsilon n})\}.

Initialize feasible set \mathcal{F}_{\beta} \leftarrow B(\mathcal{D}_c \cup \mathcal{D}_p).

for t = 1, 2, \dots do

Compute model parameters \hat{\theta} \leftarrow \operatorname{argmin}_{\theta} L\left(\theta; (\mathcal{D}_c \cup \mathcal{D}_p) \cap \mathcal{F}_{\beta}\right).

Pre-compute g_{\hat{\theta}, \mathcal{D}_{\text{test}}}^{\top} H_{\hat{\theta}}^{-1} as in (9) and (11).

for i = 1, \dots, \epsilon n do

Set \tilde{x}_i^0 \leftarrow \tilde{x}_i - \eta g_{\hat{\theta}, \mathcal{D}_{\text{test}}}^{\top} H_{\hat{\theta}}^{-1} \frac{\partial^2 \ell(\hat{\theta}; \tilde{x}_i, \tilde{y}_i)}{\partial \hat{\theta} \partial \tilde{x}_i}. (gradient update)

Set \tilde{x}_i \leftarrow \operatorname{argmin}_{x \in \mathcal{F}_{\beta}} \|x - \tilde{x}_i^0\|_2. (projection)

end for

Update feasible set \mathcal{F}_{\beta} \leftarrow B(\mathcal{D}_c \cup \mathcal{D}_p).

end for
Output \mathcal{D}_p.
```

Concentrated attacks. The influence-basic attack optimizes over each of the ϵn poisoned points separately. This has two problems: it is slow (we have to compute the gradients $\frac{\partial L}{\partial \tilde{x}}$ and do the projection step ϵn times each iteration), and the resulting poisoned points are often quite far from each other (because of differences in initialization), leaving them vulnerable to being detected as anomalies. We will show this empirically in Section 4.3.

As Theorem 1 (Section 3.1) shows, we can modify the algorithm to instead only consider copies of two distinct points $(\tilde{x}_+, 1)$ and $(\tilde{x}_-, -1)$, one from each class, without any loss in attack effectiveness. This is computationally more efficient and also results in a stronger attack. At each iteration, we only need to compute the gradients and do the projection twice (vs. ϵn times). Moreover, the resulting attack is by construction concentrated on only two distinct points, helping it evade detection.

In our implementation, we weight these two poisoned points inversely proportional to the class balance: i.e., if there are P positive and N negative points in \mathcal{D}_c , then we place $\frac{N}{P+N} \cdot \epsilon$ weight on $(\tilde{x}_+,1)$ and $\frac{P}{P+N} \cdot \epsilon$ weight on $(\tilde{x}_-,-1)$. This maintains the same class balance, on average, as the original label flip initialization.

Randomized rounding with the LP relaxation. Recall that using randomized rounding to handle integer input constraints has two issues: first, the rounded points can be far from each other (reducing concentration), and second, the rounded points might violate the constraints imposed by the L2 defense even when the continuous (unrounded) points do not. As discussed in Section 3.3, we use our linear programming relaxation and repeated points heuristic to mitigate these issues.

A final issue is that the hinge loss is piecewise linear, which means that its gradient is zero or one, and its Hessian zero, almost everywhere. As a result, the gradient of the test loss L w.r.t. \tilde{x} —which involves an inverse-Hessian-gradient-product, as in Algorithm 2—gives a very poor indication of which directions to perturb \tilde{x} , and can easily get stuck. To mitigate this problem, we follow Koh and Liang (2017) and smooth the hinge loss, replacing it with

the function $\ell_{\text{smooth}}(\theta; x, y) = \delta \log(1 + \exp(\frac{1 - y \theta^{\top} x}{\delta}))$ for some small δ . The function ℓ_{smooth} converges to ℓ as $\delta \to 0$, but has derivatives of all orders whenever $\delta > 0$.

Together, these modifications yield our improved attack, which we call the **influence** attack.

4.3 Experimental results

We tested our **influence** attack on the MNIST-1-7 and Dogfish image datasets and the Enron spam classification dataset (see Section 2.3 for details). We omitted the IMDB sentiment classification dataset because the **influence** attack on this dataset was too slow; in subsequent sections, we will introduce different attack methods that are fast enough to run on this dataset.

Recall that our goal is to create an attack \mathcal{D}_p that can increase test error regardless of which defense is deployed against it. To evaluate this, we run each defense in Section 2.2 separately against our attacks and measure the effectiveness of an attack by the minimum increase in test error it achieves over all of the defenses.

Setup. To carry out the influence attack, we alternate between taking gradient steps and re-fitting the anomaly detector parameters β , as described in Algorithm 2. We took the feasible set \mathcal{F}_{β} to be the intersection of the feasible sets under the **L2** and **slab** defenses, plus any additional input constraints that each dataset imposed, relying on the concentrated nature of the attack (i.e., only having two distinct continuous attack points) to evade the loss, **SVD**, and **k-NN** defenses. We varied the amount of poisoned data between $\epsilon = 0.5\%$ and $\epsilon = 3\%$.

For the defender, we used the hinge loss with L_2 regularization, fixing the regularization parameter via cross-validation on the clean data.³ For each defense, we set the threshold τ such that 5% of data from each class was filtered out.

Results. As shown in Figure 3, the **influence** attack significantly increases test error on the Enron dataset: with $\epsilon = 1\%$ poisoned data, test error increased from 3% to 13%, and with $\epsilon = 3\%$ poisoned data, test error increased to 21%. On the Dogfish dataset, the attack increases test error from 1% to 5% with $\epsilon = 1\%$ and 8% with $\epsilon = 3\%$. The attack is less successful on the MNIST-1-7 dataset, and does not appreciably affect test error there. These results are consistent with Steinhardt et al. (2017), which showed that the **L2** and **slab** defenses are certifiably effective at defending the MNIST-1-7 dataset, and to a lesser extent the Dogfish dataset, for low values of ϵ .

How important is each defense? Figure 3 plots one curve for each defense, representing the test error that the attacker achieves under that defense. However, our attacks are constructed to avoid all of the defenses, even though we evaluate them individually against each defense; this simulates the fact that the attacker might not know which defense will be deployed ahead of time, and therefore strives to evade all of them. We can therefore ask how much each defense is contributing to the total effectiveness of the collective defenses. In our experiments:

^{3.} In practice, the defender does not have access to the clean data, so its regularization parameter will be chosen based on the full dataset $\mathcal{D}_c \cup \mathcal{D}_p$. However, our framework assumes that the attacker knows the regularization parameter in advance. This is a potential disadvantage for the defender. In Section 7.1, we study what happens if the attacker does not know exactly how much regularization the defender will use.

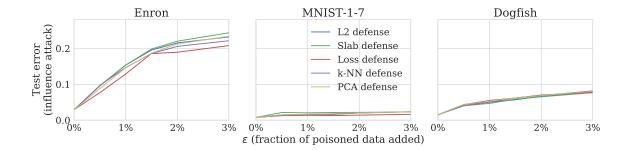


Figure 3: The **influence** attack successfully drives test error on the Enron dataset from 3% to 23% with just $\epsilon = 3\%$ poisoned data, and on the Dogfish dataset from 1% to 8%, but does not manage to significantly affect the MNIST-1-7 dataset.

- The **L2** defense is essential, as it bounds the distance that a poisoned point can be from the clean points. Without such a bound, the attacker can cause arbitrary changes to the defender's model (Biggio et al., 2012).
- The slab defense decreases the test error that the attacker achieves by a few percentage points across all of the datasets. This is particularly significant for the MNIST-1-7 and Dogfish datasets, where the overall test error is low; for example, without the slab defense, the **influence** attack achieves 4% test error on MNIST-1-7 and 8% test error on Dogfish with $\epsilon = 1\%$ poisoned data.
- The loss defense decreases the test error that the attacker achieves on the Enron dataset by a few percentage points, which we can read directly from Figure 3. This is because our randomized rounding procedure can sometimes significantly increase the loss incurred on a poisoned point (e.g., an input feature of 0.01 can sometimes be rounded up to 1, and if the coefficient for that feature is large, the rounded point could incur high loss even though the loss of the continuous, unrounded point is low).
- On the other hand, the **k-NN** and **SVD** do not contribute much to the defenses; if we removed them from consideration, our attacks would be just as effective.

To test the effect of our improvements over the **influence-basic** attack, we ran an ablative study on the Enron dataset. Comparing our attack (Figure 4-Left) to a version without the linear programming (LP) relaxation (Figure 4-Mid), we can see that the LP relaxation increases test error by a few percentage points. The difference is even more stark when comparing our attack to the **influence-basic** attack (Figure 4-Right), which does not include the LP relaxation nor the smoothed hinge or concentrated points, and consequently does not manage to increase test error beyond 10%.

Iterative optimization. Finally, we investigate the effect of iterative optimization by comparing the full attack, which iteratively updates the anomaly detector parameters β (Algorithm 1), with an attack that simply fixes β based on the clean training data at the start. (For both attacks, the defender still trains its anomaly detector on the entire training set $\mathcal{D}_c \cup \mathcal{D}_p$.) While iterative optimization does consistently slightly better (Figure 5), the

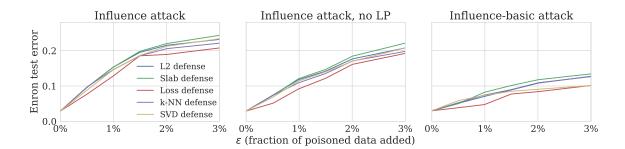


Figure 4: Ablative study of the changes we made to the **influence** attack, evaluated on the Enron dataset. Left: results of the **influence** attack. Middle: results of the **influence** attack, without the linear programming (LP) relaxation described in Section 3.3. Right: results of the **influence-basic** attack, which does not use the LP relaxation nor the smoothed hinge and concentrated attack.

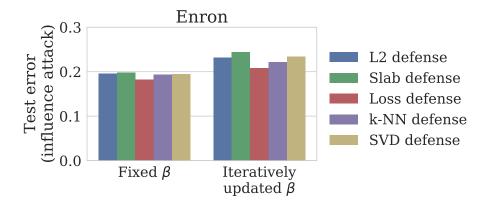


Figure 5: Iteratively updating the feasible set \mathcal{F}_{β} increases test error by a few percentage points on the Enron dataset (with $\epsilon = 3\%$ poisoned data), compared to fixing the feasible set based on just the clean data \mathcal{D}_{c} .

latter attack is still very successful; from this, at the low levels of ϵ that we choose, the attack does not shift the centroids of the data that much, and therefore the feasible set \mathcal{F}_{β} for both the **L2** and **slab** defenses stay somewhat constant.

One perspective on iterative optimization in our setting is that it is targeting the **slab** defense by trying to rotate the vector between the two class centroids; as Steinhardt et al. (2017) show, at large ϵ (e.g., $\epsilon = 0.3$, which is 10 times larger than what we consider), this vector can be significantly changed by the poisoned points, whereas the **L2** feasible set is harder to perturb. Thus, the effectiveness of iterative optimization is upper-bounded by how effective the **slab** defense is. On the Enron dataset and at the low ϵ settings that we consider, the slab defense only decreases test error by a few percentage points, so iterative optimization only increases test loss by a few percentage points. To illustrate this point, we ran an attack with $\epsilon = 0.3$ on the MNIST-1-7 dataset: the **influence** attack without

iterative optimization achieved an increase in test error of only 1.1%, while the **influence** attack with iterative optimization achieved a larger increase in test error of 7.5%.

Our conclusion is that iterative optimization only slightly increases the effectiveness of the attacks in the low ϵ setting that we focus on. For simplicity, in the rest of the paper we will therefore fix β based on the clean training data at the start of each attack and use a single instance of constrained optimization.

Comparison with related work. The influence-basic attack is equivalent to the gradient-based attacks introduced in prior work (e.g., in Biggio et al. (2012), Mei and Zhu (2015b), and Koh and Liang (2017)), except that we augment the attack by projecting back onto the feasible set (otherwise, the poisoned points will be trivially detected by the defenses). These gradient-based attacks appear to be the most popular way of carrying out indiscriminate data poisoning attacks. Our ablative analysis (Figure 4) shows that the improvements in the **influence** attack significantly increase its effectiveness against data sanitization defenses.

Steinhardt et al. (2017) successfully attack the MNIST-1-7 and Dogfish datasets using $\epsilon=0.3$, which is an order of magnitude larger than what we consider here. Their attack uses a specialized semi-definite programming formulation and relies on poisoning the anomaly detector (i.e., placing poisoned points to move the class centroids in a way that renders the slab defense ineffective). At the lower levels of ϵ that we consider in this paper, they showed that no data poisoning attack can significantly increase test error on those datasets when both the **L2** and slab defenses are used.

Note that our **influence** attacks on MNIST-1-7 at high ϵ are considerably weaker than those in Steinhardt et al. (2017), which uses a specialized semi-definite program to achieve an increase in test error of 39% for $\epsilon = 0.3$. The high- ϵ setting is not our focus in this paper, since it is less realistic; this performance gap could be a result of poor step size tuning or initialization on our part, or it could signify a weakness in the applicability of iterative optimization and/or gradient descent to the high- ϵ setting.

Discussion. The success of our attacks vary a lot from dataset to dataset. Steinhardt et al. (2017) speculate that dataset-dependent factors, like the dimensionality of the feature space and how well-separated the classes are, can significantly affect the ability of our attacks to poison the learned model.

The **influence** attack is successful against the Enron dataset, despite the additional integer constraints that it imposes. However, the drawback of the **influence** attack is that it is slow: each iteration of gradient descent requires computing an expensive inverse Hessian-vector product (10) and a projection onto the feasible set, which can also be expensive in high dimensions. This restriction makes the attack infeasible to scale to datasets with higher dimensions. Moreover, the **influence** attack relies on local optimization and can sometimes get stuck in poor local minima.

In the next two sections, we will discuss two different attack approaches that mitigate these issues.

5. KKT Attack

We propose a new attack, the **KKT** attack, that addresses two shortcomings of the **influence** attack: its computational cost and susceptibility to local minima. The **KKT** attack is based on the observation that the difficulty in the attacker's optimization problem (4) stems from how the optimization variable \mathcal{D}_p only affects the objective (test loss $L(\hat{\theta}; \mathcal{D}_{test})$) through the model parameters $\hat{\theta}$, which are themselves a complicated function of \mathcal{D}_p . In general, we do not know what $\hat{\theta}$ would lead to an attack that is both effective and realizable; but if we did know which $\hat{\theta}$ we were after, then the above optimization problem simplifies to finding \mathcal{D}_p such that $\hat{\theta} = \operatorname{argmin}_{\theta} L(\theta; \mathcal{D}_c \cup \mathcal{D}_p)$. As we will show in this section, this simplified problem can be solved much more efficiently than the original bilevel problem.

The **KKT** attack makes use of this observation by decomposing the attack into two parts: 1) using fast heuristics to find decoy parameters θ_{decoy} that we want the defender to learn, and then 2) finding poisoned data \mathcal{D}_{p} such that the defender is indeed tricked into learning the decoy parameters θ_{decoy} . The name of this attack comes from the use of the Karush-Kuhn-Tucker (KKT) first-order necessary conditions for optimality in step 2.

5.1 Attacking with known θ_{decov}

Assume for now that we have identified decoy parameters θ_{decoy} that we, as the attacker, would like the defender to learn. Our task is then to find poisoned data \mathcal{D}_p such that \mathcal{D}_p evades data sanitization and θ_{decoy} minimizes the overall training loss over both \mathcal{D}_p and the clean data \mathcal{D}_c . We can formulate this task as the following optimization problem:

find
$$\mathcal{D}_{p}$$

s.t. $|\mathcal{D}_{p}| = \epsilon |\mathcal{D}_{c}|$
 $\mathcal{D}_{p} \subseteq \mathcal{F}_{\beta}$
 $\theta_{decoy} = \underset{\theta}{\operatorname{argmin}} L(\theta; \mathcal{D}_{c} \cup \mathcal{D}_{p}).$ (12)

Since θ_{decoy} is pre-specified, we can rewrite this inner optimization as a simple equality. Specifically, if the loss ℓ is strictly convex and differentiable in θ , we can rewrite the condition

$$\begin{split} \theta_{\mathrm{decoy}} &= \operatorname*{argmin}_{\theta} L(\theta; \mathcal{D}_{\mathrm{c}} \cup \mathcal{D}_{\mathrm{p}}) \\ &= \operatorname*{argmin}_{\theta} \sum_{(x,y) \in \mathcal{D}_{\mathrm{c}}} \ell(\theta; x, y) + \sum_{(\tilde{x}, \tilde{y}) \in \mathcal{D}_{\mathrm{p}}} \ell(\theta; \tilde{x}, \tilde{y}) \end{split}$$

as the equivalent KKT optimality condition

$$\sum_{(x,y)\in\mathcal{D}_{c}} \nabla_{\theta} \ell(\theta_{\text{decoy}}; x, y) + \sum_{(\tilde{x}, \tilde{y})\in\mathcal{D}_{p}} \nabla_{\theta} \ell(\theta_{\text{decoy}}; \tilde{x}, \tilde{y}) = 0.$$
 (13)

If the loss ℓ is not differentiable, e.g., the hinge loss, we can replace this with a similar subgradient condition.

Since the first term in (13) is fixed (i.e., it does not depend on the optimization variable \mathcal{D}_{p}), we can treat it as a constant $g_{\theta_{\text{decoy}},\mathcal{D}_{c}} \stackrel{\text{def}}{=} \frac{1}{|\mathcal{D}_{c}|} \sum_{(x,y) \in \mathcal{D}_{c}} \nabla_{\theta} \ell(\theta_{\text{decoy}}; x, y)$. Rewriting

(13) as
$$g_{\theta_{\text{decoy}}, \mathcal{D}_c} + \frac{1}{|\mathcal{D}_c|} \sum_{(\tilde{x}, \tilde{y}) \in \mathcal{D}_p} \nabla_{\theta} \ell(\theta_{\text{decoy}}; \tilde{x}, \tilde{y}) = 0$$
 and substituting it into (12) gives us

find
$$\mathcal{D}_{p}$$

s.t. $|\mathcal{D}_{p}| = \epsilon |\mathcal{D}_{c}|$
 $\mathcal{D}_{p} \subseteq \mathcal{F}_{\beta}$
 $g_{\theta_{\text{decoy}},\mathcal{D}_{c}} + \frac{1}{|\mathcal{D}_{c}|} \sum_{(\tilde{x},\tilde{y})\in\mathcal{D}_{p}} \nabla_{\theta} \ell(\theta_{\text{decoy}}; \tilde{x}, \tilde{y}) = 0.$ (14)

If this optimization problem (14) has a solution, we can find it by solving the equivalent norm-minimization problem

minimize
$$\|g_{\theta_{\text{decoy}}, \mathcal{D}_{c}} + \frac{1}{|\mathcal{D}_{c}|} \sum_{(\tilde{x}, \tilde{y}) \in \mathcal{D}_{p}} \nabla_{\theta} \ell(\theta_{\text{decoy}}; \tilde{x}, \tilde{y}) \|_{2}^{2}$$
 (15)
s.t. $|\mathcal{D}_{p}| = \epsilon |\mathcal{D}_{c}|$
 $\mathcal{D}_{p} \subseteq \mathcal{F}_{\beta}$,

which moves the KKT constraint into the objective, relying on the fact that the norm of a vector is minimized when the vector is 0.

Concentrated attacks. As with the influence attack, Theorem 1 (Section 3.1) shows that we can concentrate our attacks by placing all of the poisoned points at two distinct locations without any loss in attack effectiveness; his makes our attack computationally faster and more likely to evade data sanitization. If we let $\epsilon_+ \cdot n$ and $\epsilon_- \cdot n$ be the number of positive and negative poisoned points added, respectively, we can write (15) as

$$\underset{\tilde{x}_{+},\tilde{x}_{-},\epsilon_{+},\epsilon_{-}}{\text{minimize}} \quad \left\| g_{\theta_{\text{decoy}},\mathcal{D}_{c}} + \epsilon_{+} \nabla_{\theta} \ell(\theta_{\text{decoy}}; \tilde{x}_{+}, 1) + \epsilon_{-} \nabla_{\theta} \ell(\theta_{\text{decoy}}; \tilde{x}_{-}, -1) \right\|_{2}^{2} \qquad (16)$$
s.t. $\epsilon_{+} + \epsilon_{-} = \epsilon$

$$(\tilde{x}_{+}, 1), (\tilde{x}_{-}, -1) \in \mathcal{F}_{\beta}.$$

This optimization problem is non-convex, but can be solved by local methods like gradient descent. Note that unlike the **influence** attack, this **KKT** attack does not require any expensive inverse Hessian calculations. Thus, if we have decoy parameters θ_{decoy} in mind, we can efficiently generate poisoned data \mathcal{D}_p that will make the defender learn parameters that are close to θ_{decoy} .

Evading the loss defense. Some defenses, like the loss defense in our setting, have feasible sets that depend on the model parameters $\hat{\theta}$ that the defender learns, which in turn depend on the poisoned points. This dependence makes it difficult for the attacker to explicitly constrain their poisoned points to lie within such feasible sets: the iterative optimization procedure in Section 3.2 relies on the feasible set not changing too much from one iteration to the next, but a successful attack will obtain parameters $\hat{\theta}$ that are significantly different from what they would be on clean data. f Moreover, in the case of the loss defense, iterative optimization is prone to getting stuck in a local optimum: if, at each iteration, we restrict the poisoned points to have low loss under the current value of $\hat{\theta}$, then the poisoned points will not be able to move $\hat{\theta}$ by much.

For the above reasons, we rely on concentrated attacks (Section 3.1) to evade the **loss** defense, and we do not optimize explicitly for it in the **influence** attack. While this approach is empirically effective, it relies to some extent on luck: the attacker cannot guarantee that its poisoned points will have low loss, and as we discuss in Section 4.3, the **loss** defense does decrease the test error achieved by a couple of percentage points.

In contrast, decoy parameters give attackers a computationally tractable handle on defenses like the **loss** defense. With decoy parameters, the attacker can approximately specify the feasible set \mathcal{F}_{β} independently of the learned parameters $\hat{\theta}$, since we know that if the attack works, the learned parameters $\hat{\theta}$ should be close to the decoy parameters θ_{decoy} . For example, we can handle the loss defense by adding the constraint $\ell(\theta_{\text{decoy}}; \tilde{x}, \tilde{y}) < \tau_{\tilde{y}}$ for each poisoned point $(\tilde{x}, \tilde{y}) \in \mathcal{D}_{p}$.

SVMs. In our experiments, the defender uses the hinge loss $\ell(\theta; x, y) = \max(0, 1 - y\theta^{\top}x)$ and applies ℓ_2 regularization with regularization parameter λ ((2), Section 2.2). In this setting, we can write (16) as the following:

$$\underset{\tilde{x}_{+},\tilde{x}_{-},\epsilon_{+},\epsilon_{-}}{\text{minimize}} \quad \|g_{\theta_{\text{decoy}},\mathcal{D}_{c}} - \epsilon_{+}\tilde{x}_{+} + \epsilon_{-}\tilde{x}_{-} + \lambda\theta_{\text{decoy}}\|_{2}^{2}
\text{s.t.} \quad 1 - \theta^{\top}\tilde{x}_{+} \ge 0
1 + \theta^{\top}\tilde{x}_{+} \ge 0
\epsilon_{+} + \epsilon_{-} = \epsilon
(\tilde{x}_{+},1), (\tilde{x}_{-},-1) \in \mathcal{F}_{\beta},$$
(17)

where the first two constraints ensure that $(\tilde{x}_+, 1)$ and $(\tilde{x}_-, -1)$ are both support vectors. This problem is convex when the feasible set \mathcal{F}_{β} is convex and ϵ_+ and ϵ_- are fixed; since \mathcal{F}_{β} is convex in our setting, we can grid search over ϵ_+ and ϵ_- and call a generic solver for the resulting convex problem. Pseudocode is given in Algorithm 3.

Algorithm 3 KKT attack with grid search.

```
Input: clean data set \mathcal{D}_c, feasible set \mathcal{F}_{\beta}, poisoning fraction \epsilon, grid search size T.

for t = 0, ..., T do

Set \epsilon_+ \leftarrow t\epsilon/T, \epsilon_- \leftarrow \epsilon - \epsilon_+.

Obtain \hat{\theta}, \tilde{x}_+, and \tilde{x}_- by solving (17) with fixed values of \epsilon_+, \epsilon_-.

Evaluate test error \mathcal{L}(\hat{\theta}).

end for

Pick \tilde{x}_+, \tilde{x}_-, \epsilon_+, \epsilon_- corresponding to highest test error \mathcal{L}(\hat{\theta}) found in grid search.

Output \mathcal{D}_p = \{\epsilon_+ |\mathcal{D}_c| \text{ copies of } \tilde{x}_+ \text{ and } \epsilon_- |\mathcal{D}_c| \text{ copies of } \tilde{x}_-\}.
```

5.2 Finding decoy parameters θ_{decoy}

The above section relied on having pre-specified decoy parameters θ_{decoy} . How can we efficiently find candidate decoy parameters?

Good decoy parameters, from the perspective of the attacker, should have high test error while still being achievable by some poisoned data \mathcal{D}_p . Decoy parameters that have a high loss on the clean data \mathcal{D}_c are unlikely to be achievable by an ϵ fraction of poisoned data \mathcal{D}_p ,

since it is likely that there exist other parameters that would have a lower training loss (and would therefore be learned by the defender).

Algorithm 4 Finding decoy parameters θ_{decoy}

```
Input: clean data set \mathcal{D}_c, loss threshold \gamma, number of repeats r. \theta_c \leftarrow \operatorname{argmin}_{\theta} L(\theta; \mathcal{D}_c)
\mathcal{D}_{\text{flip}} \leftarrow r \text{ copies of } \{(x, y) \in \mathcal{D}_{\text{test}} : \ell(\theta_c; x, y) \geq \gamma\}
\mathcal{D}_{\text{decoy}} \leftarrow \mathcal{D}_c \cup \mathcal{D}_{\text{flip}}
\theta_{\text{decoy}} \leftarrow \operatorname{argmin}_{\theta} L(\theta; \mathcal{D}_{\text{decoy}})
Output \mathcal{D}_{\text{decoy}}.
```

One heuristic for finding such decoy parameters is by augmenting the clean data \mathcal{D}_c with a dataset \mathcal{D}_{flip} comprising label-flipped examples from \mathcal{D}_{test} , and finding the parameters that minimize the training loss on this modified training set $\mathcal{D}_{decoy} = \mathcal{D}_c \cup \mathcal{D}_{flip}$ (Algorithm 4). The idea is that since the decoy parameters θ_{decoy} were trained on \mathcal{D}_{decoy} , which incorporates flipped points from \mathcal{D}_{test} , it might achieve high test loss. At the same time, the following informal argument suggests that the decoy parameters θ_{decoy} are likely to have low loss on the clean data \mathcal{D}_c . By construction,

$$\begin{split} |\mathcal{D}_{c}| \cdot L(\theta_{decoy}; \mathcal{D}_{c}) + |\mathcal{D}_{flip}| \cdot L(\theta_{decoy}; \mathcal{D}_{flip}) &= |\mathcal{D}_{decoy}| \cdot L(\theta_{decoy}; \mathcal{D}_{decoy}) \\ &\leq |\mathcal{D}_{decoy}| \cdot L(\theta_{c}; \mathcal{D}_{decoy}) \\ &= |\mathcal{D}_{c}| \cdot L(\theta_{c}; \mathcal{D}_{c}) + |\mathcal{D}_{flip}| \cdot L(\theta_{c}; \mathcal{D}_{flip}). \end{split}$$

Rearranging terms, this implies that

$$\begin{split} L(\theta_{\text{decoy}}; \mathcal{D}_{\text{c}}) &\leq L(\theta_{c}; \mathcal{D}_{\text{c}}) + \frac{|\mathcal{D}_{\text{flip}}|}{|\mathcal{D}_{\text{c}}|} \cdot (L(\theta_{c}; \mathcal{D}_{\text{flip}}) - L(\theta_{\text{decoy}}; \mathcal{D}_{\text{flip}})) \\ &\leq L(\theta_{c}; \mathcal{D}_{\text{c}}) + \frac{|\mathcal{D}_{\text{flip}}|}{|\mathcal{D}_{\text{c}}|} \cdot L(\theta_{c}; \mathcal{D}_{\text{flip}}), \end{split}$$

where the last inequality comes from the non-negativity of the loss L. The second term on the right-hand side, $L(\theta_c; \mathcal{D}_{\text{flip}})$, is likely to be small: $\mathcal{D}_{\text{flip}}$ comprises of points that originally had a high loss under θ_c before their labels were flipped (which implies that their label-flipped versions are likely to have a lower loss), and we can choose $|\mathcal{D}_{\text{flip}}|$ to be small compared to $|\mathcal{D}_{\text{c}}|$. Thus, the average loss $L(\theta_{\text{decoy}}; \mathcal{D}_{\text{c}})$ of the decoy parameters θ_{decoy} on the clean data \mathcal{D}_{c} is not likely to be too much higher than $L(\theta_c; \mathcal{D}_{\text{c}})$, which is the best possible average loss on \mathcal{D}_{c} within the model family.

By varying the loss threshold γ and number of repeats r used in Algorithm 4, we obtain different candidates for θ_{decoy} . Since finding an attack \mathcal{D}_{p} for each candidate θ_{decoy} is fast, our attack algorithm generates a set of candidate decoy parameters and tries all of them, picking the θ_{decoy} that achieves the highest test loss.

5.3 Experimental results

We tested our **KKT** attack against the Enron dataset, from before, as well as the larger IMDB dataset (see Table 1 for dataset details). We focus on these two datasets as they are more vulnerable to attack (compared to the MNIST-1-7 and Dogfish datasets, which have some certificates of defensibility; see Section 4 and Steinhardt et al. (2017)).

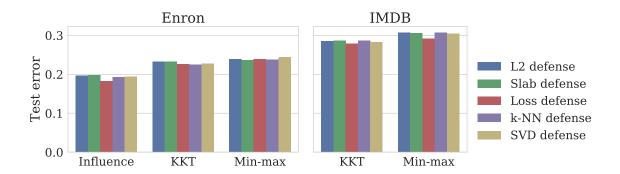


Figure 6: The **KKT** and **min-max** attacks give slightly higher test error than the **influence** attack. (Note there is stochasticity in results due to randomized rounding.) Moreover, these attacks are efficient enough to run on the IMDB dataset. Here, we show results with $\epsilon = 3\%$ poisoned data and without iterative optimization.

Setup. For each dataset, we first generated candidate decoy parameters as in Section 5.2—adding r copies of each test point whose flipped label has loss greater than γ . For Enron we swept over $r \in \{1, 2, 3, 5, 8, 12, 18, 25, 33\}$ and γ set to the qth quantile of the loss (over the flipped test set), for $q \in \{0.05, 0.10, \dots, 0.55\}$. This yielded 99 candidates θ_{decoy} ; for efficiency we removed all parameters that had lower test error and higher training loss than some other candidate θ'_{decoy} . This left us with 48 parameters total. For IMDB we applied a similar procedure but took $r \in \{1, 2, 3, 4, 5\}$ and $q \in \{0.1, 0.2, \dots, 0.6\}$; this yielded 18 candidates after pruning (we sought fewer candidates for IMDB because it is bigger and slower to attack).

We set the fraction of poisoned data to be $\epsilon = 3\%$, and for each set of decoy parameters, we grid searched over 7 different ratios of positive vs. negative poisoned points, ranging from $\epsilon_+ = 3\%$, $\epsilon_- = 0\%$ to $\epsilon_+ = 0\%$, $\epsilon_- = 3\%$. For this attack, we used the intersection of the feasible sets under the **L2**, **slab**, and **loss** defenses (since it is easy to specify the latter with decoy parameters), and relied on attack concentration to evade the **SVD** and **k-NN** defenses. We initialized the feasible set parameters β on the clean data \mathcal{D}_c and fixed them.

For the defender, as before, we set the threshold for each defense such that 5% of data from each class was filtered out, and used a ℓ_2 regularized hinge loss for training. As always, the defender trains its anomaly detector on the entire dataset $\mathcal{D}_c \cup \mathcal{D}_p$.

Results. The KKT attack achieves slightly higher test error than the influence attack on the Enron dataset (22.5% vs. 18.3%, taking the minimum over all defenses, and both without iterative optimization; Figure 6-Left). The computational efficiency of the KKT attack also allowed us to run it against the IMDB dataset—which has 6x the data points and 17x the features of the Enron dataset—achieving a test error of 27.8% (Figure 6-Right).

In our experiments, successful attacks did not need to exactly reach the decoy parameters; in fact, trying to get to ambitious (i.e., high test error but unattainable) decoy parameters could sometimes outperform exactly reaching unambitious decoy parameters. (Of course, the ideal choice of decoy parameters would have high test error but also be attainable.)

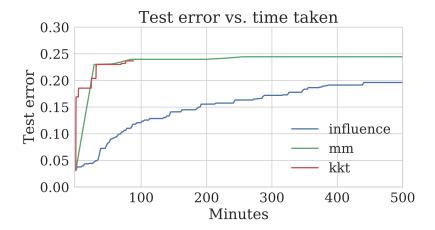


Figure 7: The test error achieved by the different attacks (against the **L2** defense), vs. the number of minutes taken to generate the attacks. Each step increase in test error represents the processing of one choice of decoy parameters (for the **KKT** and **min-max** attacks) or 10 gradient steps (for the **influence** attack).

Timing. We measured the speed of the **KKT** and **influence** attacks against the Enron dataset by running them each on 2 cores of an Intel Xeon E5 processor. Additionally, the **influence** attack used a GeForce GTX Titan X for GPU-based calculations. Despite not using a GPU, the **KKT** attack was significantly faster: to reach 17% error, the **KKT** attack took 27 seconds while the **influence** attack took 286 minutes (Figure 7).

Discussion. The **KKT** attack is efficient and effective when suitable decoy parameters can be generated. Its speed, relative to the **influence** attack, come from two sources: it avoids expensive inverse Hessian-vector product calculations (10); and since it is solving a convex problem (as opposed to the non-convex problem that the **influence** attack is doing gradient descent on), it admits efficient general-purpose convex optimization solvers.

However, the **KKT** attack has some drawbacks:

- 1. The **KKT** attack relies on the convexity of the optimization problem (17), which is a stronger condition than generally requiring that the model loss be convex. The **influence** attack is more generally applicable.
- 2. Our current method of generating decoy parameters (Algorithm 4) does not take into account any defenses, so it could fail at generating good decoy parameters that are achievable under more sophisticated defenses.
- 3. Finally, as with the **influence** attack, our current implementation of the **KKT** attack requires grid-searching over the ratio of positive to negative points. This quickly becomes infeasible in multi-class settings (though gradient descent on the class balance could still be effective).

Note that in some settings, the goal of the attacker can be to make the defender learn some particular target parameters, as opposed to our current attacker goal of increasing test loss (Mei and Zhu, 2015b). The **KKT** attack offers a direct way of attacking in such situations: set θ_{decoy} to the attacker's target parameters. This can be significantly more efficient, and less vulnerable to getting stuck in local minima, than the variants of the **influence** attack that have been used in previous work (Mei and Zhu, 2015b).

6. Improved Min-Max Attack

Our third and final attack is the **min-max** attack, which improves on what we call the **min-max-basic** attack from prior work. Our **min-max** attack relies on the same decoy parameters introduced in Section 5, but unlike the **influence** and **KKT** attacks, it naturally handles multi-class problems and does not require convexity of the feasible set. Its drawback is assuming that the clean data \mathcal{D}_c and the test data \mathcal{D}_{test} are drawn from the same distribution (i.e., that the attacker is performing an *indiscriminate* attack, see Sections 2 and 8).

We start by reviewing the **min-max-basic** attack, as it was introduced in Steinhardt et al. (2017).

6.1 The min-max-basic attack

Recall that the attacker's goal is to find poisoned points \mathcal{D}_p that maximize the test loss $L(\hat{\theta}; \mathcal{D}_{test})$ that the defender incurs, where the parameters $\hat{\theta}$ are chosen to minimize the training loss $L(\hat{\theta}; \mathcal{D}_c \cup \mathcal{D}_p)$ (equation (4)). As we discussed in Sections 3.2, 4, and 5, the bilevel nature of this optimization problem—maximizing the loss involves an inner minimization to find the parameters $\hat{\theta}$ —makes it difficult to solve.

The key insight in Steinhardt et al. (2017) was that we can make this problem tractable by replacing the test loss $L(\hat{\theta}; \mathcal{D}_{\text{test}})$ with the training loss $L(\hat{\theta}; \mathcal{D}_{\text{c}} \cup \mathcal{D}_{\text{p}})$. This substitution changes the bilevel problem into a saddle-point problem—i.e., one that can be expressed in the form $\min_{u} \max_{v} f(u, v)$ — that can be solved efficiently via gradient descent.

To do so, we first approximate the average test loss with the average clean training loss:

$$L(\theta; \mathcal{D}_{\text{test}}) \approx L(\theta; \mathcal{D}_{\text{c}}).$$
 (18)

This approximation only works in the setting where the test data $\mathcal{D}_{\text{test}}$ is drawn from the same distribution as the (clean) training data \mathcal{D}_{c} , and relies on the training set being sufficiently big and the model being appropriately regularized, such that test loss is similar to training loss. Next, we make use of the non-negativity of the loss to upper bound the average clean training loss with the average combined loss on the clean and poisoned data:

$$L(\theta; \mathcal{D}_{c}) \leq L(\theta; \mathcal{D}_{c}) + \epsilon L(\theta; \mathcal{D}_{p}) = (1 + \epsilon)L(\theta; \mathcal{D}_{c} \cup \mathcal{D}_{p}), \tag{19}$$

where, as usual, ϵ is the relative ratio of poisoned points $\epsilon = \frac{|\mathcal{D}_p|}{|\mathcal{D}_c|}$.

By combining (18) and (19), we can approximately upper-bound the average test loss $L(\theta; \mathcal{D}_{test})$ by $(1 + \epsilon)$ times the average loss on the combined training data $L(\theta; \mathcal{D}_c \cup \mathcal{D}_p)$. Instead of directly optimizing for $L(\theta; \mathcal{D}_{test})$ as the attacker (equation (4)), we can therefore

optimize for $L(\theta; \mathcal{D}_c \cup \mathcal{D}_p)$, which gives us

$$\begin{split} & \underset{\mathcal{D}_{p} \subseteq \mathcal{F}_{\beta}}{\text{maximize}} & & L(\theta; \mathcal{D}_{c} \cup \mathcal{D}_{p}) \\ & \text{where} & & \hat{\theta} \overset{\text{def}}{=} \underset{\theta}{\text{argmin}} L(\theta; \mathcal{D}_{c} \cup \mathcal{D}_{p}). \end{split}$$

The advantage of this formulation is that the outer maximization and inner minimization are over the same function $L(\theta; \mathcal{D}_c \cup \mathcal{D}_p)$, which lets us rewrite it as the saddle-point problem

$$\max_{\mathcal{D}_{p} \subseteq \mathcal{F}_{\beta}} \min_{\theta} L(\hat{\theta}; \mathcal{D}_{c} \cup \mathcal{D}_{p}). \tag{20}$$

When the loss ℓ is convex, we can solve (20) by swapping min and max and solving the resulting problem $\min_{\theta} \max_{\mathcal{D}_p \subset \mathcal{F}_{\theta}} L(\theta; \mathcal{D}_c \cup \mathcal{D}_p)$, which expands out to

$$\min_{\theta} \left[\frac{\lambda(1+\epsilon)}{2} \|\theta\|_{2}^{2} + \frac{1}{|\mathcal{D}_{c}|} \sum_{(x,y)\in\mathcal{D}_{c}} \ell(\theta;x,y) + \epsilon \max_{(\tilde{x},\tilde{y})\in\mathcal{F}_{\beta}} \ell(\theta;\tilde{x},\tilde{y}) \right]. \tag{21}$$

This problem is convex when the loss ℓ is convex, and we can solve it via (sub-)gradient descent by iteratively finding $(\tilde{x}, \tilde{y}) \in \mathcal{F}_{\beta}$ that maximizes $\ell(\theta; \tilde{x}, \tilde{y})$, then taking the gradient of the outer expression w.r.t. (\tilde{x}, \tilde{y}) .

To solve the inner problem of finding $(\tilde{x}, \tilde{y}) \in \mathcal{F}_{\beta}$ that maximizes $\ell(\theta; \tilde{x}, \tilde{y})$, we note that if the model is margin-based, i.e., $\ell(\theta; \tilde{x}, \tilde{y}) = c(-y\theta^{\top}x)$ for some monotone increasing function c (which is the case for SVMs and logistic regression), then maximizing $\ell(\theta; \tilde{x}, \tilde{y})$ is equivalent to minimizing the margin $y\theta^{\top}x$. For a fixed \tilde{y} , we can solve the convex problem

$$\begin{array}{ll}
\text{minimize} & \tilde{y}\theta^{\top}\tilde{x} \\
\text{s.t.} & (\tilde{x}, \tilde{y}) \in \mathcal{F}_{\beta}.
\end{array}$$

To find the $(\tilde{x}, \tilde{y}) \in \mathcal{F}_{\beta}$ that maximizes the loss $\ell(\theta; \tilde{x}, \tilde{y})$, we therefore enumerate over the possible choices of \tilde{y} , solving the above convex problem for each \tilde{y} , and pick the one that gives the smallest (most negative) margin.

Steinhardt et al. (2017) show that if (21) is minimized incrementally via ϵn iterations of gradient descent, then we can form a strong attack \mathcal{D}_p out of the corresponding set of ϵn maximizers $\{(\tilde{x}, \tilde{y})\}$. Pseudocode is given in Algorithm 5.

This algorithm automatically handles class balance, since at each iteration it chooses to add either a positive or negative point; it can thus handle multi-class attacks without additional difficulty, unlike the **KKT** or **influence** attacks. Moreover, unlike the **influence** attack, it avoids solving the expensive bilevel optimization problem, which makes it fast enough to run on larger datasets like the IMDB sentiment dataset.

6.2 Improvements to the basic algorithm

In this paper, we improve the **min-max-basic** attack by incorporating the decoy parameters introduced in Section 5 (Algorithm 4).

The **min-max-basic** attack aims to maximize the combined training loss $L(\hat{\theta}; \mathcal{D}_c \cup \mathcal{D}_p)$, and achieves this by repeatedly adding in the highest-loss point that lies in the feasible

Algorithm 5 min-max-basic attack.

```
Input: clean data \mathcal{D}_{c}, poisoned fraction \epsilon, burn-in n_{burn}, feasible set \mathcal{F}_{\beta}.

Initialize: \theta \in \mathbf{R}^{d}, \mathcal{D}_{p} \leftarrow \emptyset.

for t = 1, \dots, n_{burn} + \epsilon n do

Pick(\tilde{x}, \tilde{y}) \in \operatorname{argmax}_{(x,y) \in \mathcal{F}_{\beta}} \ell(\theta; x, y). \qquad \text{(Find highest-loss point in } \mathcal{F}_{\beta}\text{)}
\theta \leftarrow \theta - \eta(\lambda \theta + \nabla L(\theta) + \epsilon \nabla \ell(\theta; \tilde{x}, \tilde{y})) \qquad \text{(Gradient update)}
if t > n_{burn} then
\mathcal{D}_{p} \leftarrow \mathcal{D}_{p} \cup \{(\tilde{x}, \tilde{y})\} \qquad \text{(Add point to attack set)}
end if
end for
Output \mathcal{D}_{p}.
```

set \mathcal{F}_{β} . The problem with this approach is that at low ϵ , the attack might end up picking poisoned points \mathcal{D}_{p} that are not fit well by the model, i.e., with high $L(\hat{\theta}; \mathcal{D}_{p})$. These points could still lead to a high combined loss $L(\hat{\theta}; \mathcal{D}_{c} \cup \mathcal{D}_{p})$, but such an attack would be ineffective for two reasons:

- 1. The loss on the clean data $L(\hat{\theta}; \mathcal{D}_c)$ might still be low, implying that the test loss would also be low. Such a scenario could happen if there is no model that fits both the poisoned points \mathcal{D}_p and the clean points \mathcal{D}_c well, since ϵ is small, overall training loss could then be minimized by fitting \mathcal{D}_c well at the expense of \mathcal{D}_p .
- 2. If the poisoned points have high loss compared to the clean points, they are likely to be filtered by the **loss** defense.

We therefore want to keep the loss on the poisoned points, $L(\hat{\theta}; \mathcal{D}_p)$, small. To do so, we use the *decoy parameters* θ_{decoy} from Section 5 (Algorithm 4), augmenting the feasible set \mathcal{F}_{β} with the constraint

$$\ell(\theta_{\text{decov}}; x, y) \le \tau, \tag{22}$$

for some fixed threshold τ . At each iteration, the attacker thus searches for poisoned points (\tilde{x}, \tilde{y}) that maximize loss under the current parameters θ while having low loss under the decoy parameters θ_{decoy} . This procedure addresses the two issues above:

- 1. Adding poisoned points with high loss under the current parameters but low loss under the decoy parameters θ_{decoy} is likely to drive the learned parameters towards θ_{decoy} . In turn, this will increase the test loss, since θ_{decoy} is chosen to have high test loss Section 5.2.
- 2. If the learned parameters are driven towards θ_{decoy} , the poisoned points in \mathcal{D}_{p} will have low loss due to the constraint (22), and hence will not get filtered by the **loss** defense.

We find that empirically, the **min-max-basic** attack naturally yields attacks that are quite concentrated. For datasets with integer input constraints, we additionally use the linear programming relaxation and repeated points heuristic (Section 3.3). Altogether, these changes to the **min-max-basic** attack yield what we call the **min-max** attack.

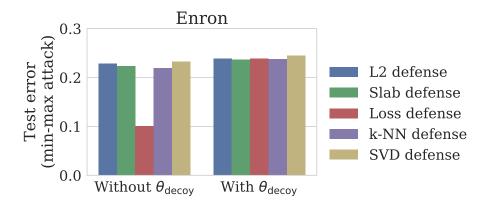


Figure 8: The use of decoy parameters allows the **min-max** attack to evade the **loss** defense.

6.3 Experimental results

Setup. To test the **min-max** attack, we followed the same experimental setup as the **KKT** attack (Section 5): using the Enron and IMDB datasets, running without alternating optimization, and with the same set of decoy parameters. To evade the loss defense, we used the threshold $\tau=0.25$ across all experiments. The results were fairly robust to this choice of threshold.

Results. The min-max attack successfully poisoned the Enron and IMDB datasets, achieving test errors of 23.7% and 29.2% respectively (Figure 6). This is comparable to the effectiveness of the KKT attack, and slightly better than the influence attack. In terms of computational efficiency, the min-max attack took 28 minutes to process its first decoy parameter, which gave 23.0% error; this is slower than the KKT attack, but significantly faster than the influence attack Figure 7.

To measure the effect of using decoy parameters in the **min-max** attack, we ran the **min-max-basic** attack from Steinhardt et al. (2017), augmented with the linear programming relaxation and repeated points heuristic. (The unaugmented version of the **min-max-basic** attack from Steinhardt et al. (2017) performs worse.) In contrast to the **min-max** attack, the **min-max-basic** attack gets defeated by the **loss** defense (test error 10.0%, Figure 8); without the constraints imposed by the decoy parameters, the poisoned points found by the **min-max-basic** attack have high loss, as discussed in Section 6.2.

Multi-class. One advantage of the **min-max** attack is that it works even when the input domain $\mathcal{X} \times \mathcal{Y}$ is discrete or the feasible set $\mathcal{F}_{\beta} \subset \mathcal{X} \times \mathcal{Y}$ is non-convex, so long as we can still efficiently solve $\max_{(x,y)\in\mathcal{F}_{\beta}}\ell(\theta;x,y)$. In particular, the **min-max** attack can search over different choices of \tilde{y} for each poisoned points (\tilde{x},\tilde{y}) , whereas the **influence** and **KKT** attacks require us to grid search over the relative proportion of the poisoned points, a strategy that quickly becomes infeasible in multi-class problems. Indeed, we need k(k-1) distinct points to carry out any data poisoning attack against a k-class SVM (Proposition 10 in Appendix A), so the grid search would take time exponential in k^2 .

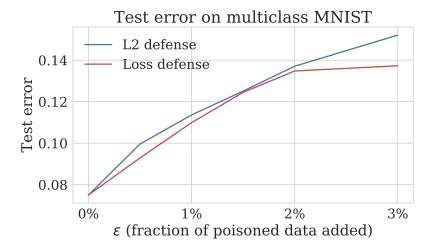


Figure 9: The **min-max** attack scales to handle multi-class attacks, as it automatically chooses the class balance / relative proportion of poisoned points. Here, we show that the **min-max** attack can increase test error on the MNIST dataset to 13.7% with $\epsilon=3\%$ poisoned data.

We illustrate a multi-class attack by running the **min-max** attack on the 10-class MNIST dataset (LeCun et al., 1998). We use the multi-class SVM formulation in Crammer and Singer (2002) as a classifier and train the model using AdaGrad (Duchi et al., 2010), with a batch size of 20, step size $\eta = 0.02$, and 3 passes over the training data. Using $\epsilon = 3\%$ poisoned data, the **min-max** attack obtains 15.2% test error against the **L2** defense and 13.7% test error against the **loss** defense, demonstrating a high-leverage attack in a multi-class setting.

Discussion. As the preceding paragraph shows, the **min-max** attack effectively handles multi-class problems, whereas the **influence** and **KKT** attacks would require an exponential grid search. (One might be able to adapt the latter attacks to automatically tune the relative weight of points, but that is a non-trivial extension which we leave for future work.)

The drawback of the **min-max** attack is that it only works when the test and training data are drawn from the same distribution, since it uses the training loss as a proxy for the test loss. In contrast, the **influence** and **KKT** do not make any assumptions on the nature of the test data.

Steinhardt et al. (2017) use the **min-max-basic** attack to get upper bounds on how effective any attack can be against the **L2** and **slab** defenses. We discuss this more in the related work (Section 8).

7. Transferability

In Sections 4-6, we saw that the **influence**, **KKT**, and **min-max** attacks are all effective on the Enron and IMDB datasets if the attacker knows the exact model that the defender is using. We summarize the relative merits of these attacks in Table 2.

Our final set of experiments study the question of *transferability*—are these attacks still effective when the attacker does not have exact knowledge of the defender, as in often the

Attack	Enron	Time	Pros and cons	
	test error	taken		
Influence	18.3%	286min	No need to find θ_{decoy}	
			Slow	
			Less transferable	
KKT	22.5%	27s	θ_{decoy} handles loss defense (but need to find good θ_{decoy})	
Min-max	23.7%	28min	θ_{decoy} handles loss defense (but need to find good θ_{decoy})	
			Handles multi-class setting	
			Assumes indiscriminate attack	

Table 2: Comparison of attacks. Enron test error is reported at $\epsilon = 3\%$, and the time taken is how long each attack took to reach 17% Enron test error.

case in practice? To model this, we use the same attacks on the Enron dataset as described in the above sections, but vary the defender's choice of 1) the amount of regularization, 2) optimization algorithm, and 3) loss function. This creates a mismatch between what the attacker thinks the defender is doing and what the defender actually does.

In general, our attacks are still effective under these changes, with the **min-max** attack generally being the most robust and the **influence** attack being the least. However, the **loss** defense poses problems for all three attacks when the optimization algorithm or the loss function are changed, suggesting that attackers should set conservative thresholds against the **loss** defense.

7.1 Regularization

Do attacks that are optimized for one level of regularization still work well at other levels of regularization? Recall that the defender uses L_2 regularization with the hyperparameter λ controlling the amount of regularization (Equation 2); in particular, we use $\lambda = 0.09$ for the Enron dataset (Table 1). To test the effect of the defender's choice of λ , we varied it from 0.009 to 0.9 while keeping the attacker's λ constant at 0.09.

Figure 10 shows the results:

- Against the **L2** defense, test error generally increased with L_2 regularization strength (Figure 10-Left). Note that the **L2** feasible set depends only on the location of the poisoned points and does not depend on the model parameters that the defender learns. Thus, the change in test error is due to the defender learning a different set of parameters given exactly the same set of poisoned points; it is not due to a change in which poisoned points are being filtered out by the defenses.
- The amount of regularization had different effects on each attack's effectiveness against the **L2** defense. In particular, the **influence** attack for a constant attacker regularization became less effective as we reduced defender regularization (< 10% test error against the **L2** defense at $\lambda = 0.009$), while the **min-max** attack was robust to changes in defender regularization (22% test error at $\lambda = 0.009$).

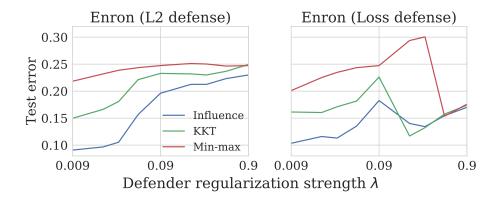


Figure 10: The effect of changing the defender's regularization strength on the test error achieved by each attack. The attacker's regularization is fixed at $\lambda = 0.09$.

- Increasing regularization can make the **loss** defense more effective (Figure 10-Right). The **loss** feasible set is sensitive to changes in the model parameters that the defender learns, so some poisoned points that evaded this defense under the original model (with $\lambda = 0.09$) are now detected under the changed model.
- Unlike the other attacks, the **min-max** attack initially gets more effective against the **loss** defense as defender regularization is increased from $\lambda = 0.09$. We suspect that this is due to the **min-max** attack using a fixed loss threshold τ (see equation (22) in Section 6.2) that is more conservative than the **KKT** attack (which uses an adaptive threshold based on the quantiles of the loss under the decoy parameters) and the **influence** attack (which solely relies on concentrated attacks to overcome the **loss** defense).

These results imply that attackers should optimize for lower levels of regularization, just in case the defender goes to a lower level (and conversely, it suggests that defenders might want to use lower levels of regularization than they might otherwise). Attackers using decoy parameters might also decide to set their loss thresholds more conservatively (i.e., choose poisoned points that obtain a lower loss under the learned parameters).

These results also suggest that our data poisoning attacks are not exploiting overfitting. If that were the case, we would expect that increasing regularization would decrease overfitting and thus reduce the effectiveness of the attacks. Instead, we observe the opposite in practice: the defender suffers when increasing regularization (although it may be able to filter out more poisoned points via the **loss** defense). Intuitively, a larger amount of regularization makes it harder for the defender's model to fit both the poisoned training points and clean training points well, and if the clean training points are not fit well, the test error will consequently increase.

7.2 Optimization and loss function

In our previous experiments, we assumed that the defender would learn the model parameters $\hat{\theta}$ that globally minimized training loss. In practice, defenders might use stochastic optimization and/or early stopping, leading to parameters $\hat{\theta}$ that are close to but not exactly at the optimum.

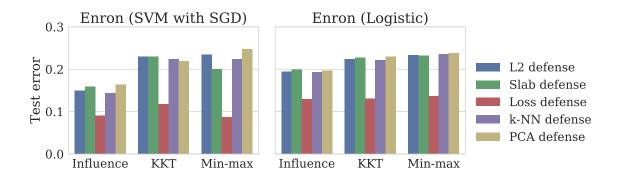


Figure 11: With the significant exception of the **loss** defense, the attack results are robust to shifts in the optimization algorithm (from an exact solution to a single pass of stochastic gradient descent) and the loss function (from the hinge loss to the logistic loss).

Figure 11-Left shows the results of our attacks on a defender that learned a model by doing a single pass of stochastic gradient descent over the dataset (i.e., each sample is looked at exactly once). We also tested how an attacker using the hinge loss would fare against a defender who uses the logistic loss (Figure 11-Right). All three attacks stayed effective against all of the defenses except the **loss** defense, which managed to significantly reduce the damage inflicted by the attacker.

As in Section 7.1, these results suggest that attackers should use a conservative loss threshold to harden their attacks against loss-based defenses. It also suggests that the attacker's ability to evade loss-based defenses is especially sensitive (relative to other defenses) to getting the defender's model correct.

8. Related Work

In this section, we discuss other attack settings and defense strategies that have been studied in the literature. For broad surveys on this topic, see e.g., Barreno et al. (2010), Biggio et al. (2014), Gardiner and Nagaraja (2016), and Papernot et al. (2016b).

8.1 Attacks

Gradient-based attacks. The **influence** attack that we presented in Section 4 is a gradient-based attack that iteratively modifies each attack point $(\tilde{x}, \tilde{y}) \in \mathcal{D}_p$ by following the gradient of the test loss with respect to \tilde{x} . This type of gradient-based data poisoning attack was first studied in the context of SVMs by Biggio et al. (2012), and has subsequently been extended to linear and logistic regression (Mei and Zhu, 2015b), topic modeling (Mei and Zhu, 2015a), collaborative filtering (Li et al., 2016), and neural networks (Koh and Liang, 2017; Yang et al., 2017; Muñoz-González et al., 2017).

Label-flip attacks. All of the attacks presented in this work control both the label \tilde{y} and input features \tilde{x} of the poisoned points. Attacks that instead only control the label \tilde{y} are

known as label-flip attacks. In a typical label-flip attack, the attacker gets to change some ϵ fraction of the labels y of the training set but is unable to change the features x (Biggio et al., 2011; Xiao et al., 2012, 2015). We experimented with a variant of the label flip attack described in Xiao et al. (2012), where we allowed the attacker to pick examples from the test set, flip their labels, and add them to the training set (Appendix B). We found that this attack, though fast to run, was significantly less effective than our proposed attacks; control of \tilde{x} seems to be necessary to carry out high-leverage attacks in the presence of data sanitization defenses.

Targeted vs. indiscriminate attacks. In all of our experiments, the attacker sought to increase error on a test set $\mathcal{D}_{\text{test}}$ that was drawn from the same distribution as the clean training data \mathcal{D}_{c} . This type of attack is known as an *indiscriminate* attack (Barreno et al., 2010), and is akin to a denial-of-service attack.

Indiscriminate attacks seek to change the predictions of the learned model on a good fraction of the entire data distribution, and therefore require substantial changes to the model. This makes indiscriminate attacks statistically interesting, as they get at fundamental properties of the model: how might an attacker that only controls 1% of the training data bring about a 10% increase in test error?

A different type of attack is a targeted attack, in which the attacker seeks to cause errors on specific test examples or small sub-populations of test examples (Gu et al., 2017; Chen et al., 2017; Burkard and Lagesse, 2017; Koh and Liang, 2017; Shafahi et al., 2018; Suciu et al., 2018). For example, an attacker might seek to have all of the emails that they send marked as non-spam while leaving other emails unaffected. Targeted attackers only seek to change the predictions of the model on a small number of instances, and therefore might be able to add in 50 poisoned training points to cause an error on a single test point (Shafahi et al., 2018). Targeted attacks are well-motivated from a security perspective: attackers might only care about a subset of the model's prediction, and targeted attacks require less control over the training set and are therefore easier to carry out.

The **influence** and **KKT** attacks in Sections 4 and 5 do not make any assumptions on the nature of the test set \mathcal{D}_{test} , and can therefore handle the targeted attack setting without modification. In contrast, the **min-max** attack in Section 6 assumes that the training error is a good approximation to the test error, and thus is only appropriate in the indiscriminate attack setting.

Backdoor attacks. A backdoor attack is a targeted attack that seeks to cause examples that contain a specific backdoor pattern, e.g., a bright sticker (Gu et al., 2017) or a particular type of sunglasses (Chen et al., 2017), to be misclassified. Backdoor attacks work by superimposing the chosen backdoor pattern onto particular training examples from a given class, which causes the model to associate the backdoor pattern with that class (and therefore misclassify, at test time, examples of a different class that also contain the backdoor pattern). The attackers in Gu et al. (2017) and Chen et al. (2017) do not need to know the model that the defender is using; in fact, the attacker in Chen et al. (2017) does not even need any knowledge of the training set, instead adding examples from a external dataset. These weaker assumptions on attacker capabilities are common in targeted attacks. In contrast, the indiscriminate attacks that we develop in this paper make use of knowledge of the model and the training set in order to have high leverage.

Clean-label attacks. Clean-label attacks are attacks that "do not require control over the labeling function; the poisoned training data appear to be labeled correctly according to an expert observer" (Shafahi et al., 2018). Not requiring control over labeling makes it easier for the attacker to practically conduct such an attack, as the attacker only needs to introduce the unlabeled poisoned data into the general pool of data (e.g., uploading poisoned images or sending poisoned emails, as Shafahi et al. (2018) discusses) and wait for the defender to label and ingest the poisoned data.

The backdoor attacks discussed above are examples of clean-label attacks, provided the backdoor pattern is chosen to be innocuous enough to avoid human suspicion. Another way of constructing clean-label attacks is by constraining the poisoned points to be close, in some metric, to a clean training point of the same class; Shafahi et al. (2018) does this with the ℓ_2 norm, while Suciu et al. (2018) and Koh and Liang (2017) use the ℓ_{∞} norm.

Our attacks are not clean-label attacks, in that the poisoned points will not necessarily be labeled as the correct class by a human expert. On the other hand, our poisoned points are designed to evade detection by automatic outlier detectors; note that "clean-label" points can fool human experts but still look like statistical outliers.

Adversarial examples and test-time attacks. The bulk of recent research in machine learning security has focused on test-time attacks, where the attacker perturbs the test example to obtain a desired classification, leaving the training data and the model unchanged. This line of research was sparked by the striking discovery that test images could be perturbed in a visually-imperceptible way and yet fool state-of-the-art neural network image classifiers (Szegedy et al., 2014; Goodfellow et al., 2015; Carlini et al., 2016; Kurakin et al., 2016; Papernot et al., 2016a, 2017; Moosavi-Dezfooli et al., 2016). Designing models that are robust to such attacks, as well as coming up with more effective attacks, is an active area of research (Papernot et al., 2016c; Madry et al., 2017; Tramèr et al., 2017; Wong and Kolter, 2018; Raghunathan et al., 2018; Sinha et al., 2018; Athalye et al., 2018; Papernot and McDaniel, 2018).

In contrast to these test-time attacks, data poisoning attacks are train-time attacks: the attacker leaves the test example unchanged, and instead perturbs the training data so as to affect the learned model. Data poisoning is less well-studied, and compared to test-time attacks, it is harder to both attack and defend in the data poisoning setting: data poisoning attacks have to depend on the entire training set, whereas test-time attacks only depend on the learned parameters; and common defense techniques against test-time attacks, such as 'adversarial training' (Goodfellow et al., 2015), do not have analogues in the data poisoning setting.

8.2 Defenses

Less work has been done on how to defend against data poisoning attacks. In the literature, effective defenses typically require additional information than the defenders we consider in this paper, e.g., having a labeled set of outliers or having a trusted dataset.

Using labeled outlier data and other training metadata. If the defender has access to data that has been labeled as 'normal' vs. 'outlier', then outlier detection can be treated as a standard supervised classification problem (Hodge and Austin, 2004). For example,

an online retailer might have a set of transactions that had been previously labeled as fraudulent, and could train a separate outlier detection model to detect and throw out other fraudulent-looking transactions from the dataset. One drawback of this method is that in an adversarial setting there is no assumption that future poisoned points might look like previous poisoned points. Such methods are therefore more suited for detecting outliers caused by natural noise processes rather than adversaries.

Instead of directly using 'normal' vs. 'outlier' labels, defenders can instead rely on other types of training metadata. For example, Cretu et al. (2008)—which introduced the term 'data sanitization' in the context of machine learning— uses information on the time at which each training point was added to the training set. The intuition is that "in a training set spanning a sufficiently large time interval, an attack or an abnormality will appear only in small and relatively confined time intervals" (Cretu et al., 2008).

Using trusted data. Other defense methods rely on having a trusted subset \mathcal{T} of the training data that only contains clean data (obtained for example by human curation). One example is the Reject on Negative Impact (RONI) defense proposed by Nelson et al. (2008), which was one of the first papers studying data poisoning attacks and defenses. The RONI defense iterates over training points (x, y) and rejects points if the model learned on just the trusted data \mathcal{T} is significantly different from the model learned on $\mathcal{T} \cup \{(x, y)\}$. Another example is the outlier detector introduced in Paudice et al. (2018), which operates similarly to our \mathbf{k} -NN defense except that it measures distances only to the points in the trusted subset.

Having a trusted dataset makes it easier for the defender, though such a dataset might be expensive or even impossible to collect; if the defender has enough resources to collect a large amount of trusted data, then they can train a model on only the trusted data, solving the problem of data poisoning. The question of whether a small amount of trusted data is sufficient for defeating attackers while maintaining high overall performance (i.e., not rejecting clean training points that are not similar to the trusted data) is an open one. Finally, defenses that rely on trusted data are particularly vulnerable to attackers that manage to compromise the trusted data (e.g., through clean-label attacks that escape human notice).

High-dimensional robust estimation. Recent work in the theoretical computer science community studies robust estimators in high dimensions, which seek to work well even in the presence of outliers. A key issue is that many traditional robust estimators incur a \sqrt{d} increase in error in d dimensions. This theoretical insight aligns with the empirical results in this paper showing that it is often possible to attack classifiers with only a small fraction of poisoned data.

Motivated by these issues, Klivans et al. (2009) and later Awasthi et al. (2014) and Diakonikolas et al. (2017b) design robust classification algorithms that avoid the poor dimension-dependence of traditional estimators, although only under strong distributional assumptions such as log-concavity. Separately, Lai et al. (2016) and Diakonikolas et al. (2016) designed robust procedures for mean estimation, which again required strong distributional assumptions. Later work (Charikar et al., 2017; Diakonikolas et al., 2017a; Steinhardt et al., 2018) showed how to perform mean estimation under much more mild assumptions, and Diakonikolas et al. (2017a) showed that their procedure could yield robust estimates in

practice. In recent concurrent work, Diakonikolas et al. (2018) showed that robust estimation techniques can be adapted to classification and used this to design a practical algorithm that appears more robust than many traditional alternatives. It would be interesting future work to attack this latter algorithm in order to better vet its robustness.

Certified defenses. Steinhardt et al. (2017) explore the task of provably certifying defenses, i.e., computing a dataset-dependent upper bound on the maximum test loss that an attacker can cause the defender to incur. Their method—from which we adopt the min-max-basic attack, in the present work—shows that the L2 and slab defenses are sufficient for defending models trained on the MNIST-1-7 and Dogfish datasets but cannot certifiably protect models trained on the Enron and IMDB datasets, which matches with the experimental results in Sections 4-6. Open questions are whether our improvements to the min-max-basic (e.g., decoy parameters) can be used in their framework to derive tighter upper bounds on attack effectiveness, and whether the other defenses (e.g., the loss defense) can be incorporated into their framework.

9. Discussion

Data poisoning attacks pose risks to machine learning systems deployed in the real world. To better understand our vulnerabilities and ability to defend against these attacks, we systematically studied how data poisoning attacks fare against a broad range of data sanitization defenses.

We were able to develop three distinct attacks that could evade data sanitization defenses while significantly increasing test error on the Enron and IMDB datasets. The **influence** attack directly optimizes for increasing the test loss through gradient ascent on the poisoned points; the **KKT** attack chooses poisoned points to achieve pre-determined decoy parameters; and the **min-max** attack efficiently solves for the poisoned points that maximize train loss, as a proxy for test loss. Compared to previous data poisoning attacks, e.g., the gradient-based attacks in Biggio et al. (2012), Mei and Zhu (2015b), and Koh and Liang (2017); the min-max attack in Steinhardt et al. (2017); or the label flip attacks in Xiao et al. (2012) (described in Section 8 and Appendix B), our attacks significantly increase test error even in the presence of data sanitization defenses.

Our conclusion is that simple data sanitization defenses fail. We need to develop better defenses to effectively counter data poisoning attacks, and as our results above show, these defenses need to be tested against attackers that are specifically geared to evade them. We end by highlighting future directions of study arising from our work.

What makes datasets and models attackable? The effectiveness of our attacks vary significantly from dataset to dataset (e.g., linear models trained on the Enron and IMDB datasets are more vulnerable than linear models trained on the MNIST-1-7 and Dogfish datasets). What conditions make certain models on certain datasets attackable, but not others? This is an open question; we speculate that linear models on the Enron and IMDB datasets are easier to attack because they have higher dimensionality and are less linearly separable, but this question deserves more study.

Non-convex models. One limitation of our attacks is that they rely on the attacker and defender being able to find the model parameters θ that globally minimize the training loss.

This is a reasonable assumption if the loss is convex, but most models in practice today (e.g., neural networks) are non-convex. Analysis of attack algorithms in the non-convex setting is substantially more difficult, e.g., a poisoning attack that works against a neural net trained with a given random seed might fail when the random seed is changed, or a single poisoned point might cause the defender to get stuck at a very bad local minimum. The targeted attack algorithms discussed in Section 8 (Gu et al., 2017; Chen et al., 2017; Shafahi et al., 2018; Suciu et al., 2018) circumvent this difficulty by adding poisoned points that resemble the test point but have a different label, at the cost of lower leverage. Open questions include whether our attacks, which change the global minimizer, remain effective in the non-convex setting, and whether the methods used for targeted attacks can be adapted to design high-leverage indiscriminate attacks against non-convex models.

Strategies for stronger defenses. What prospects are there for building stronger defenses that are robust against determined attackers? We outline several approaches, as well as potential difficulties.

One strategy would be to try to design better outlier detectors—perhaps the **L2** and **slab** defenses provide too crude a measure of whether a point is realistic, and sophisticated generative models such as generative adversarial nets (Goodfellow et al., 2014) could better rule out poisoned data. We are skeptical of this approach, as there are many natural distributions (such as multivariate Gaussians) where even a perfect generative model cannot prevent an adversary from substantially skewing empirical statistics of the data (see Steinhardt (2018), Section 1.3). The existence of adversarial test examples for image classifiers (Szegedy et al., 2014; Goodfellow et al., 2015) also weights against this approach, since such examples are generated using a method for inducing high loss under a target model. This method could likely be adapted for use in the **min-max** attack, as the the main sub-routine in that attack involves generating examples that induce high loss under a target model.

Another strategy rests on the observation that if we could directly minimize the 0-1 test error (rather than using a convex proxy), then an adversary controlling an ϵ -fraction of the data could always induce at most additional $\frac{\epsilon}{1-\epsilon}$ error, at least on the training set.⁴ The key issue with convex proxies for the 0/1-loss is that they are unbounded and so an adversary can cause the loss on \mathcal{D}_p to be very large. Perhaps one can do better by instead using non-convex but bounded proxies for the 0/1-loss? This would make optimization of the training loss more difficult, but might pay off with higher robustness. It unfortunately also opens up a new avenue for attack—the attacker could try to push the learner towards a bad local minimum. There are also known hardness results for even approximately minimizing the 0/1-loss (Feldman et al., 2009; Guruswami and Raghavendra, 2009), but it is possible that they do not apply in practice.

Finally, as noted in Section 8, there is recent work seeking to design estimators that are *provably robust* to adversarial training data under suitable distributional assumptions. Diakonikolas et al. (2018) recently presented a practical implementation of these methods for classification and regression tasks and showed promising initial results. We view provable security under a well-defined threat model as the gold standard, and encourage further

^{4.} To see this, note that the 0/1-loss of θ^* averaged across $\mathcal{D}_c \cup \mathcal{D}_p$ is at most at most ϵ larger than across \mathcal{D}_c , so any $\hat{\theta}$ outperforming θ^* can only have slightly higher loss than $\hat{\theta}$ across \mathcal{D}_c .

work along this direction (we refer the interested reader to Li (2018) or Steinhardt (2018) for two recent overviews). There appears to be plenty of room both to improve the practical implementation of this family of defenses, and to devise new theoretically-grounded procedures.

Reproducibility

The code and data for replicating our experiments are available on GitHub (https://github.com/kohpangwei/data-poisoning-journal-release).

Acknowledgments

We are grateful to Steve Mussmann, Zhenghao Chen, Marc Rasi, and Robin Jia for helpful comments and discussion. This work was partially funded by an Open Philanthropy Project Award. PWK was supported by the Facebook Fellowship Program. JS was supported by the Fannie and John Hertz Foundation Fellowship.

References

- N. Agarwal, B. Bullins, and E. Hazan. Second order stochastic optimization in linear time. arXiv preprint arXiv:1602.03943, 2016.
- A. Athalye, N. Carlini, and D. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. arXiv preprint arXiv:1802.00420, 2018.
- P. Awasthi, M. F. Balcan, and P. M. Long. The power of localization for efficiently learning linear separators with noise. In *Symposium on Theory of Computing (STOC)*, pages 449–458, 2014.
- I. Bárány and R. Karasev. Notes about the Carathéodory number. Discrete & Computational Geometry, 48(3):783–792, 2012.
- J. F. Bard. Some properties of the bilevel programming problem. *Journal of optimization theory and applications*, 68(2):371–378, 1991.
- J. F. Bard. Practical Bilevel Optimization: Algorithms and Applications. Springer, 1999.
- M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.
- B. Biggio, B. Nelson, and P. Laskov. Support vector machines under adversarial label noise. ACML, 20:97–112, 2011.
- B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. In *International Conference on Machine Learning (ICML)*, pages 1467–1474, 2012.
- B. Biggio, G. Fumera, and F. Roli. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):984–996, 2014.

STRONGER DATA POISONING

- C. Burkard and B. Lagesse. Analysis of causative attacks against SVMs learning from data streams. In *International Workshop on Security And Privacy Analytics*, 2017.
- N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou. Hidden voice commands. In *USENIX Security*, 2016.
- M. Charikar, J. Steinhardt, and G. Valiant. Learning from untrusted data. In *Symposium on Theory of Computing (STOC)*, 2017.
- X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint arXiv:1712.05526, 2017.
- K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine learning*, 47(2):201–233, 2002.
- G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *IEEE Symposium on Security* and Privacy, pages 81–95, 2008.
- I. Diakonikolas, G. Kamath, D. Kane, J. Li, A. Moitra, and A. Stewart. Robust estimators in high dimensions without the computational intractability. In *Foundations of Computer Science (FOCS)*, 2016.
- I. Diakonikolas, G. Kamath, D. Kane, J. Li, A. Moitra, and A. Stewart. Being robust (in high dimensions) can be practical. *arXiv*, 2017a.
- I. Diakonikolas, D. M. Kane, and A. Stewart. Learning geometric concepts with nasty noise. arXiv, 2017b.
- I. Diakonikolas, G. Kamath, D. M. Kane, J. Li, J. Steinhardt, and A. Stewart. Sever: A robust meta-algorithm for stochastic optimization. arXiv preprint arXiv:1803.02815, 2018.
- J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *Conference on Learning Theory (COLT)*, 2010.
- V. Feldman, P. Gopalan, S. Khot, and A. K. Ponnuswami. On agnostic learning of parities, monomials, and halfspaces. *SIAM Journal on Computing*, 39(2):606–645, 2009.
- J. Gardiner and S. Nagaraja. On the security of machine learning in malware C&C detection: A survey. ACM Computing Surveys (CSUR), 49(3), 2016.
- I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Advances in Neural Information Processing Systems (NIPS), 2014.
- I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015.
- T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. arXiv preprint arXiv:1708.06733, 2017.

- V. Guruswami and P. Raghavendra. Hardness of learning halfspaces with noise. SIAM Journal on Computing, 39(2):742–765, 2009.
- V. Hodge and J. Austin. A survey of outlier detection methodologies. Artificial intelligence review, 22(2):85–126, 2004.
- A. R. Klivans, P. M. Long, and R. A. Servedio. Learning halfspaces with malicious noise. Journal of Machine Learning Research (JMLR), 10:2715–2740, 2009.
- P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning (ICML)*, 2017.
- A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. arXiv, 2016.
- K. A. Lai, A. B. Rao, and S. Vempala. Agnostic estimation of mean and covariance. In Foundations of Computer Science (FOCS), 2016.
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- B. Li, Y. Wang, A. Singh, and Y. Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. In Advances in Neural Information Processing Systems (NIPS), 2016.
- J. Li. Principled Approaches to Robust Machine Learning and Beyond. PhD thesis, Massachusetts Institute of Technology, 2018.
- A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In Association for Computational Linguistics (ACL), 2011.
- A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks (published at ICLR 2018). *arXiv*, 2017.
- J. Martens. Deep learning via hessian-free optimization. In *International Conference on Machine Learning (ICML)*, pages 735–742, 2010.
- S. Mei and X. Zhu. The security of latent Dirichlet allocation. In *Artificial Intelligence and Statistics (AISTATS)*, 2015a.
- S. Mei and X. Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In Association for the Advancement of Artificial Intelligence (AAAI), 2015b.
- V. Metsis, I. Androutsopoulos, and G. Paliouras. Spam filtering with naive Bayes which naive Bayes? In *CEAS*, volume 17, pages 28–69, 2006.
- V. Mirrokni, R. P. Leme, A. Vladu, and S. C. Wong. Tight bounds for approximate Carathéodory and beyond. arXiv preprint arXiv:1512.08602, 2015.

- S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2016.
- L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, pages 27–38, 2017.
- B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. A. Sutton, J. D. Tygar, and K. Xia. Exploiting machine learning to subvert your spam filter. *LEET*, 8:1–9, 2008.
- A. Newell, R. Potharaju, L. Xiang, and C. Nita-Rotaru. On the practicality of integrity attacks on document-level sentiment analysis. In Workshop on Artificial Intelligence and Security (AISec), pages 83–93, 2014.
- N. Papernot and P. McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. arXiv preprint arXiv:1803.04765, 2018.
- N. Papernot, P. McDaniel, and I. Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv*, 2016a.
- N. Papernot, P. McDaniel, A. Sinha, and M. Wellman. Towards the science of security and privacy in machine learning. *arXiv*, 2016b.
- N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security* and Privacy, pages 582–597, 2016c.
- N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, B. Z. Celik, and A. Swami. Practical black-box attacks against machine learning. In *Asia Conference on Computer and Communications Security*, pages 506–519, 2017.
- A. Paudice, L. Muñoz-González, A. Gyorgy, and E. C. Lupu. Detection of adversarial training examples in poisoning attacks through anomaly detection. arXiv preprint arXiv:1802.03041, 2018.
- B. A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1): 147–160, 1994.
- A. Raghunathan, J. Steinhardt, and P. Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2018.
- B. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S. Lau, S. Rao, N. Taft, and J. Tygar. Antidote: Understanding and defending against poisoning of anomaly detectors. In ACM SIGCOMM Conference on Internet measurement conference, 2009.
- A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison Frogs! Targeted clean-label poisoning attacks on neural networks. arXiv preprint arXiv:1804.00792, 2018.

- A. Sinha, H. Namkoong, and J. Duchi. Certifiable distributional robustness with principled adversarial training. In *International Conference on Learning Representations (ICLR)*, 2018.
- J. Steinhardt. Robust Learning: Information Theory and Algorithms. PhD thesis, Stanford University, 2018.
- J. Steinhardt, P. W. Koh, and P. Liang. Certified defenses for data poisoning attacks. In Advances in Neural Information Processing Systems (NIPS), 2017.
- J. Steinhardt, M. Charikar, and G. Valiant. Resilience: A criterion for learning in the presence of arbitrary outliers. In *Innovations in Theoretical Computer Science (ITCS)*, 2018.
- O. Suciu, R. Mărginean, Y. Kaya, H. D. III, and T. Dumitraş. When does machine learning fail? generalized transferability for evasion and poisoning attacks. arXiv preprint arXiv:1803.06975, 2018.
- C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.
- F. Tramèr, A. Kurakin, N. Papernot, D. Boneh, and P. McDaniel. Ensemble adversarial training: Attacks and defenses. arXiv preprint arXiv:1705.07204, 2017.
- E. Wong and J. Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning (ICML)*, 2018.
- H. Xiao, H. Xiao, and C. Eckert. Adversarial label flips attack on support vector machines. In European Conference on Artificial Intelligence, 2012.
- H. Xiao, B. Biggio, B. Nelson, H. Xiao, C. Eckert, and F. Roli. Support vector machines under adversarial label contamination. *Neurocomputing*, 160:53–62, 2015.
- C. Yang, Q. Wu, H. Li, and Y. Chen. Generative poisoning attack method against neural networks. *arXiv*, 2017.

Appendix A. How many distinct points are needed for data poisoning attacks?

Consider some attack \mathcal{D}_p which makes a defender learn model parameters θ . Under what conditions does there exist some other attack \mathcal{D}'_p that contains at most as many points $(|\mathcal{D}_p| \geq |\mathcal{D}'_p|)$, but with fewer distinct points (i.e., \mathcal{D}'_p contains repeated copies of points)?

If the attacker could place poisoned points at arbitrary locations, and if the model's loss function is unbounded (as is the case in most models, e.g., SVMs or logistic regression), then very few poisoned points (distinct or otherwise) are generally needed since the attacker

can get high leverage over the model by placing a poisoned point far away. However, in our setting, the attacker is constrained to play poisoned points that are in the feasible set \mathcal{F} .

In this section, we provide a general method for finding the minimum number of distinct points necessary for achieving any attack, given a model with a strictly convex loss function and some feasible set \mathcal{F} . We show that for binary SVMs and logistic regression, if \mathcal{F} is convex for each class—as is the case for the **L2**, slab, loss, and SVD—then only 2 distinct points are necessary.

As a high-level sketch, our proof proceeds as follows:

- 1. (Proposition 4) We consider the set of scaled feasible gradients $\mathcal{G}_{\hat{\theta}} \stackrel{\text{def}}{=} \{\alpha \nabla_{\theta} \ell(\hat{\theta}; x, y) : 0 \leq \alpha \leq 1, (x, y) \in \mathcal{F}\}$. This set corresponds to the gradients of all feasible points, scaled by some $0 \leq \alpha \leq 1$. We show that the number of distinct points needed for an attack relates to the geometry of this set $\mathcal{G}_{\hat{\theta}}$. In particular, if $\mathcal{G}_{\hat{\theta}}$ is convex for each class, then only 2 points are needed.
- 2. (Proposition 7) We check that for SVMs, $\mathcal{G}_{\hat{\theta}}$ is convex for each class if the original feasible set \mathcal{F} is convex for each class.
- 3. (Proposition 8) More generally, we establish conditions under which differentiable margin-based losses have $\mathcal{G}_{\hat{\theta}}$ convex for each class, and we show that logistic regression satisfies these conditions.

For convenience, in the sequel we will assume that these models are trained by finding

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{arg\,min}} \quad \frac{\lambda}{2} \|\theta\|_2^2 + \sum_{(x,y) \in \mathcal{D}_c} \ell(\theta; x, y) + \sum_{(\tilde{x}, \tilde{y}) \in \mathcal{D}_p} \ell(\theta; \tilde{x}, \tilde{y}). \tag{23}$$

In other words, the degree of regularization is not explicitly affected by the total number of data points $|\mathcal{D}_c| + |\mathcal{D}_p|$. Moreover, the overall loss is strictly convex due to regularization, even if $\ell(\theta; x, y)$ is only convex (and not strictly convex) in θ , as is the case with the hinge loss. We also assume that $\mathcal{D}_p \subseteq \mathcal{F}$ (otherwise, the poisoned points will simply be thrown out by the defender).

We start by establishing the equivalence between the number of distinct points needed to poison a given model and the geometry of the set of feasible gradients of that model.

Definition 3 The Carathéodory number of a set $\mathcal{G} \subseteq \mathbb{R}^n$ is the smallest number c such that each $\tilde{g} \in conv(\mathcal{G})$ can be written as a convex combination of at most c points in \mathcal{G} . (Each \tilde{g} may be a convex combination of a different set of c points.)

Proposition 4 Consider a defender who learns a model by first discarding all points outside a fixed feasible set \mathcal{F} , and then finding the parameters that minimize a strictly convex loss $\ell(\theta; x, y)$ averaged over the training set. If a parameter $\hat{\theta}$ is attainable by any set of \tilde{n} poisoned points $\mathcal{D}_p = \{(\tilde{x}_1, \tilde{y}_1), \dots, (\tilde{x}_{\tilde{n}}, \tilde{y}_{\tilde{n}})\} \subseteq \mathcal{F}$, then there exists a set $\tilde{\mathcal{D}}_p$ that also attains $\hat{\theta}$ with at most \tilde{n} poisoned points but only contains c distinct points (with a potentially fractional number of repeats of each point), where c is the Carathéodory number of the set of possible scaled gradients $\mathcal{G}_{\hat{\theta}} \stackrel{\text{def}}{=} \{\alpha \nabla_{\theta} \ell(\hat{\theta}; x, y) : 0 \leq \alpha \leq 1, (x, y) \in \mathcal{F}\}$.

Remark 5 If $\ell(\theta; x, y)$ is not differentiable in θ , we obtain an equivalent result by considering the subgradient sets $\partial_{\theta}\ell$. In this case, we can define $\mathcal{G}_{\hat{\theta}} \stackrel{\text{def}}{=} \{\alpha g : 0 \leq \alpha \leq 1, g \in \bigcup_{(x,y)\in\mathcal{F}} \partial_{\theta}\ell(\hat{\theta}; x, y)\}$. For clarity in the following proof, we will assume that $\ell(\theta; x, y)$ is differentiable, but the argument for the non-differentiable case is almost identical.

Proof Assume that we are given a set of n clean training points \mathcal{D}_c and a set of \tilde{n} poisoned points \mathcal{D}_p . Without loss of generality, we assume that each poisoned point $(\tilde{x}, \tilde{y}) \in \mathcal{D}_p$ lies in the feasible set \mathcal{F} (otherwise, the poisoned point will be filtered out and have no effect).

The defender learns parameters $\hat{\theta}$ that minimize the training loss

$$\frac{\lambda}{2} \|\theta\|_2^2 + \sum_{(x,y)\in\mathcal{D}_c} \ell(\theta; x, y) + \sum_{(\tilde{x}, \tilde{y})\in\mathcal{D}_D} \ell(\theta; \tilde{x}, \tilde{y}). \tag{24}$$

Since $\hat{\theta}$ is a minimum of the loss, we have that

$$0 = \lambda \hat{\theta} + \sum_{(x,y)\in\mathcal{D}_{c}} \nabla_{\theta} \ell(\hat{\theta}; x, y) + \sum_{(\tilde{x}, \tilde{y})\in\mathcal{D}_{p}} \nabla_{\theta} \ell(\hat{\theta}; \tilde{x}, \tilde{y}), \tag{25}$$

where $\nabla_{\theta}\ell(\hat{\theta};x,y)$ denotes the gradient of the loss at the point (x,y) with parameters $\hat{\theta}$.

Our goal is to find the minimum number of distinct points c such that given any clean dataset \mathcal{D}_c , attack \mathcal{D}_p , and consequent model parameters $\hat{\theta}$, we can find some other attack $\tilde{\mathcal{D}}_p$ with at most c distinct points such that $\tilde{\mathcal{D}}_p$ also makes the defender learn $\hat{\theta}$. The key observation is that because the loss is strictly convex (by assumption), (25) is a necessary and sufficient condition for the defender to learn $\hat{\theta}$. In particular, if we define $g \stackrel{\text{def}}{=} \sum_{(\tilde{x},\tilde{y})\in\mathcal{D}_p} \nabla_{\theta}\ell(\hat{\theta};\tilde{x},\tilde{y})$, then any other attack $\tilde{\mathcal{D}}_p$ that satisfies $g = \sum_{(\tilde{x},\tilde{y})\in\tilde{\mathcal{D}}_p} \nabla_{\theta}\ell(\hat{\theta};\tilde{x},\tilde{y})$ —note that we have $\tilde{\mathcal{D}}_p$ in place of \mathcal{D}_p —will also make the defender learn $\hat{\theta}$.

What values can g take on? Since $g \stackrel{\text{def}}{=} \sum_{(\tilde{x},\tilde{y}) \in \mathcal{D}_p} \nabla_{\theta} \ell(\hat{\theta}; \tilde{x}, \tilde{y})$ by construction, and each point (\tilde{x}, \tilde{y}) in \mathcal{D}_p lies in the feasible set \mathcal{F} , we know that the normalized vector $\tilde{g} \stackrel{\text{def}}{=} g/\tilde{n}$ (where \tilde{n} is the number of points in $\tilde{\mathcal{D}}_p$) is contained in the convex hull $\text{conv}(\mathcal{G}_{\hat{\theta}})$, where $\mathcal{G}_{\hat{\theta}}$ is the set of scaled gradients that are possible in the feasible set, $\mathcal{G}_{\hat{\theta}} \stackrel{\text{def}}{=} \{\alpha \nabla_{\theta} \ell(\hat{\theta}; x, y) : 0 \leq \alpha \leq 1, (x, y) \in \mathcal{F}\}$.

Now, say we can find c points $g_1, g_2, \ldots, g_c \in \mathcal{G}_{\hat{\theta}}$ such that \tilde{g} is a convex combination of these points (i.e., $\tilde{g} \in \gamma_1 g_1 + \gamma_2 g_2 + \ldots + \gamma_k g_c$, with $\gamma_i \geq 0$ and $\sum_i \gamma_i = 1$). Since each point $g_i \in \mathcal{G}_{\hat{\theta}}$ can be written as $\alpha_i \nabla_{\theta} \ell(\hat{\theta}; \tilde{x}_i, \tilde{y}_i)$ for some $(\tilde{x}_i, \tilde{y}_i) \in \mathcal{F}$ with $0 \leq \alpha_i \leq 1$, we can construct a dataset $\tilde{\mathcal{D}}_p$ comprising $\alpha_1 \gamma_1 \tilde{n}$ copies of $(\tilde{x}_1, \tilde{y}_1)$, $\alpha_2 \gamma_2 \tilde{n}$ copies of $(\tilde{x}_2, \tilde{y}_2)$, and so on, such that $g = \tilde{n}\tilde{g} = \tilde{n}\sum_{i=1}^c \alpha_i \gamma_i \nabla_{\theta} \ell(\hat{\theta}; \tilde{x}_i, \tilde{y}_i)$. This constructed dataset $\tilde{\mathcal{D}}_p$ will therefore attain $\hat{\theta}$ with only c distinct points, and since $\sum_{i=1}^c \alpha_i \gamma_i \leq 1$, the total weight of all of these points will be less than or equal to \tilde{n} .

We thus want to find the smallest number c such that for any $\tilde{g} \in \text{conv}(\mathcal{G}_{\hat{\theta}})$, we can write \tilde{g} as a convex combination of at most c points in $\mathcal{G}_{\hat{\theta}}$. This is the definition of the Carathéodory number of $\mathcal{G}_{\hat{\theta}}$, as desired.

^{5.} g is actually contained in the convex hull of the unscaled gradient set $\{\nabla_{\theta}\ell(\hat{\theta};x,y):(x,y)\in\mathcal{F}\}$, which is a subset of the scaled gradient set, so the above proof also goes through if we consider the unscaled gradient set in place of the scaled gradient set. However, as we will see later in this section, adding the α scaling term reduces the Carathéodory number, which gives a stronger result.

Proposition 4 tells us that to find the number of distinct points required for data poisoning attacks on a given model and feasible set, it suffices to find the Carathéodory number of the set of feasible gradients of that model. Finding the Carathéodory number of a set is a well-studied problem (see, e.g., Bárány and Karasev (2012), or Mirrokni et al. (2015) for an approximate version of the problem). In our setting, each feasible set can be written as the union of a small number of convex sets, which simplifies the analysis of its Carathéodory number. We start by establishing the following lemma:

Lemma 6 If a set \mathcal{G} is the union of k convex sets, $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2 \cup \dots \mathcal{G}_k$ where each \mathcal{G}_i is convex, then the Carathéodory number of \mathcal{G} is at most k.

Proof Pick any $\tilde{g} \in \text{conv}(\mathcal{G})$. By construction, we can write $\tilde{g} = \sum_{i=1}^k \sum_{j=1}^{n_i} \alpha_{ij} x_{ij}$ where $x_{ij} \in \mathcal{G}_i$, $\alpha_{ij} \geq 0$, and $\sum_i \sum_j \alpha_{ij} = 1$. Since the \mathcal{G}_i are convex sets, we can find $\tilde{x}_i \in \mathcal{G}_i \subseteq \mathcal{G}$ such that $\tilde{x}_i = \sum_{j=1}^{n_i} \alpha_{ij} x_{ij} / \sum_{j=1}^{n_i} \alpha_{ij}$, allowing us to write $\tilde{g} = \sum_{i=1}^k \left(\sum_{j=1}^{n_i} \alpha_{ij}\right) \tilde{x}_i$. Since any $\tilde{g} \in \text{conv}(\mathcal{G})$ can be written as the convex combination of at most k points in \mathcal{G} , the Carathéodory number of \mathcal{G} is k.

We use this lemma to establish the Carathéodory number of the set of scaled gradients for a binary SVM.

Proposition 7 (Carathéodory number of \mathcal{G} **for a 2-class SVM)** Consider the setting of Proposition 4, and let the loss function be the ℓ_2 -regularized hinge loss on data with binary labels. Suppose that for each class y = -1, +1, the feasible set $\mathcal{F}_y \stackrel{\text{def}}{=} \{x : (x, y) \in \mathcal{F}\}$ is a convex set. Then the Carathéodory number of $\mathcal{G}_{\hat{\theta}} \stackrel{\text{def}}{=} \{\alpha g : 0 \leq \alpha \leq 1, g \in \bigcup_{(x,y)\in\mathcal{F}} \partial_{\theta}\ell(\hat{\theta}; x, y)\}$ is at most 2, independent of $\hat{\theta}$.

Proof Recall that in a binary SVM, the loss on an individual point is given by

$$\ell(\theta; x, y) = \max(0, 1 - y\theta^{\mathsf{T}} x). \tag{26}$$

For convenience, we have folded the regularization term into the loss on each point.

From Proposition 4, we want to find the Carathéodory number of the set of all possible scaled (sub)gradients of poisoned points $\mathcal{G} \stackrel{\text{def}}{=} \{ \alpha g : 0 \leq \alpha \leq 1, g \in \bigcup_{(x,y)\in\mathcal{F}} \partial_{\theta} \ell(\hat{\theta}; x, y) \}$. For a binary SVM, the subgradient sets are:

$$\partial_{\theta}\ell(\hat{\theta}; x, y) = \begin{cases} \{0\}, & \text{if } y\hat{\theta}^{\top}x > 1\\ \{-\gamma yx : 0 \le \gamma \le 1\} & \text{if } y\hat{\theta}^{\top}x = 1\\ \{-yx\} & \text{if } y\hat{\theta}^{\top}x < 1. \end{cases}$$
(27)

Plugging this into the expression for $\mathcal{G}_{\hat{\theta}}$, we get that

$$\mathcal{G}_{\hat{\theta}} = \{ -\alpha yx : 0 \le \alpha \le 1, (x, y) \in \mathcal{F}, y \hat{\theta}^{\top} x \le 1 \}, \tag{28}$$

which we can rewrite as the union of two convex sets, one for each class:

$$\mathcal{G}_{\hat{\theta}} = \{ -\alpha x : 0 \le \alpha \le 1, (x, +1) \in \mathcal{F}_{+1}, \hat{\theta}^{\top} x \le 1 \} \cup \{ \alpha x : 0 \le \alpha \le 1, (x, -1) \in \mathcal{F}_{-1}, -\hat{\theta}^{\top} x \le 1 \}.$$

By Lemma 6, the Carathéodory number of $\mathcal{G}_{\hat{\theta}}$ is at most 2, regardless of what $\hat{\theta}$ is.

More generally, we can bound the Carathéodory number of the set of scaled gradients for a particular class of convex, differentiable, margin-based losses.

Proposition 8 (Carathéodory number of \mathcal{G} for margin-based losses) Consider the setting of Proposition 4 on binary data, and let the loss function be $\ell(\theta; x, y) = c(-y\theta^{\top}x)$, where $c : \mathbb{R} \to \mathbb{R}$ is a convex, monotone increasing, and twice-differentiable function. Suppose that the ratio of the second to the first derivative of c, c''/c' is a monotone non-increasing function, and that for each class y = -1, +1, the feasible set $\mathcal{F}_y \stackrel{\text{def}}{=} \{x : (x, y) \in \mathcal{F}\}$ is a convex set. Then the Carathéodory number of $\mathcal{G}_{\hat{\theta}} \stackrel{\text{def}}{=} \{\alpha \nabla_{\theta} \ell(\hat{\theta}; x, y) : 0 \le \alpha \le 1, (x, y) \in \mathcal{F}\}$ is at most 2, independent of $\hat{\theta}$.

Proof We can write $\mathcal{G}_{\hat{\theta}}$ as the union of two sets $\mathcal{G}_{\hat{\theta},+1}$ and $\mathcal{G}_{\hat{\theta},-1}$, one for each class, where $\mathcal{G}_{\hat{\theta},y} \stackrel{\text{def}}{=} \{\alpha \nabla_{\theta} \ell(\hat{\theta};x,y) : 0 \leq \alpha \leq 1, x \in \mathcal{F}_y\}$. To show that the Carathéodory number of $\mathcal{G}_{\hat{\theta}}$ is at most 2, it suffices to show that each class set $\mathcal{G}_{\hat{\theta},y}$ is convex and then apply Lemma 6.

Pick $\hat{\theta}$ arbitrarily and fix y to be -1 or +1. To check that $\mathcal{G}_{\hat{\theta},y}$ is convex, it suffices to check that for each possible choice of x_1 and x_2 in \mathcal{F}_y and $0 \le \gamma \le 1$, there exists some $\tilde{x} \in \mathcal{F}_y$ and $0 \le \alpha \le 1$ such that

$$\alpha \nabla_{\theta} \ell(\hat{\theta}; \tilde{x}, y) = \gamma \nabla_{\theta} \ell(\hat{\theta}; x_1, y) + (1 - \gamma) \nabla_{\theta} \ell(\hat{\theta}; x_2, y). \tag{29}$$

Since our loss function has the form $\ell(\theta; x, y) = c(-y\theta^{\top}x)$, we have that

$$\nabla_{\theta} \ell(\hat{\theta}; x, y) = c'(-y\theta^{\top} x)(-yx), \tag{30}$$

where c' is the derivative of c. Substituting this into (29) and cancelling out the -y terms on both sides gives us the equivalent condition

$$\alpha c'(-y\theta^{\top}\tilde{x})\tilde{x} = \gamma c'(-y\theta^{\top}x_1)x_1 + (1-\gamma)c'(-y\theta^{\top}x_2)x_2. \tag{31}$$

To satisfy the above condition, we will take

$$\tilde{x} = \frac{\gamma c'(-y\theta^{\top}x_1)}{\gamma c'(-y\theta^{\top}x_1) + (1-\gamma)c'(-y\theta^{\top}x_2)} x_1 + \frac{(1-\gamma)c'(-y\theta^{\top}x_2)}{\gamma c'(-y\theta^{\top}x_1) + (1-\gamma)c'(-y\theta^{\top}x_2)} x_2, \quad (32)$$

which in turn implies that

$$\alpha = \frac{\gamma c'(-y\theta^{\top}x_1) + (1-\gamma)c'(-y\theta^{\top}x_2)}{c'(-y\theta^{\top}\tilde{x})}.$$
(33)

Since \tilde{x} is a convex combination of x_1 and x_2 , the convexity of \mathcal{F}_y implies that $\tilde{x} \in \mathcal{F}_y$, so it remains to check that $0 \le \alpha \le 1$.⁶ Moreover, since c is monotone increasing by assumption, c' is positive, and therefore α is positive. It remains to check that $\alpha \le 1$.

First, note that $\log c'(\cdot)$ is a concave function, as its derivative c''/c' is monotone non-increasing by assumption. For notational convenience, let $s_1 \stackrel{\text{def}}{=} -y\theta^\top x_1$ and $s_2 \stackrel{\text{def}}{=} -y\theta^\top x_2$, and let $T \stackrel{\text{def}}{=} \gamma c'(s_1) + (1 - \gamma)c'(s_2)$. We then have that

$$\log c'(-y\theta^{\top}\tilde{x}) = \log c'\left(\frac{\gamma c'(s_1)s_1 + (1-\gamma)c'(s_2)s_2}{T}\right)$$

$$\geq (\gamma c'(s_1)/T)\log c'(s_1) + ((1-\gamma)c'(s_2)/T)\log c'(s_2) \qquad \text{(Jensen's)}$$

$$= (\gamma c'(s_1)/T)\log\frac{\gamma c'(s_1)/T}{\gamma} + ((1-\gamma)c'(s_2)/T)\log\frac{(1-\gamma)c'(s_2)/T}{\gamma} + \log T$$

$$\geq \log T \qquad \qquad \text{(non-negativity of KL divergence)}$$

Exponentiating both sides and rearranging gives us

$$\alpha = \frac{T}{c'(-y\theta^{\top}\tilde{x})} \le 1.$$

The above argument shows that $\mathcal{G}_{\hat{\theta},y}$ is a convex set. Since we picked $\hat{\theta}$ and y arbitrarily, we can apply Lemma 6 to conclude that $\mathcal{G}_{\hat{\theta}} = \mathcal{G}_{\hat{\theta},+1} \cup \mathcal{G}_{\hat{\theta},-1}$ has Carathéodory number at most 2 regardless of $\hat{\theta}$.

Corollary 9 (Carathéodory number of \mathcal{G} for logistic regression) Consider a logistic regression model in the setting of Proposition 8, where $\ell(\theta; x, y) = \log(1 + \exp(-y\theta^{\top}x))$. Then the Carathéodory number of $\mathcal{G}_{\hat{\theta}} \stackrel{\text{def}}{=} \{\alpha \nabla_{\theta} \ell(\hat{\theta}; x, y) : 0 \leq \alpha \leq 1, (x, y) \in \mathcal{F}\}$ is at most 2, independent of $\hat{\theta}$.

Proof In logistic regression, we have that $c(v) = \log(1 + \exp(v))$; $c'(v) = \sigma(v)$ where σ is the sigmoid function, $\sigma(v) \stackrel{\text{def}}{=} \frac{1}{1 + \exp(-v)}$; and $c''(v) = \sigma(v)(1 - \sigma(v))$. Thus, $\frac{c''}{c'}$ is a monotone decreasing function, and c is convex, monotone increasing, and twice-differentiable, so Proposition 8 applies.

We can collect all of the above results into the following theorem, which appears in the main text.

Theorem 1 (2 points suffice for 2-class SVMs and logistic regression) Consider a defender who learns a 2-class SVM or logistic regression model by first discarding all points outside a fixed feasible set \mathcal{F} , and then finding the parameters that minimize the average (regularized) training loss. Suppose that for each class y = -1, +1, the feasible set

^{6.} Note that since $\alpha c'(-y\theta^{\top}\tilde{x})$ is a scalar, so for (31) to hold, \tilde{x} needs to point in the same direction as $\gamma c'(-y\theta^{\top}x_1)x_1 + (1-\gamma)c'(-y\theta^{\top}x_2)x_2$. Moreover, since we need \tilde{x} to be in \mathcal{F}_y , we want \tilde{x} to be a convex combination of x_1 and x_2 . This choice of \tilde{x} is the only choice that satisfies these two considerations.

 $\mathcal{F}_y \stackrel{\text{def}}{=} \{x : (x,y) \in \mathcal{F}\}\$ is a convex set. If a parameter $\hat{\theta}$ is attainable by any set of \tilde{n} poisoned points $\mathcal{D}_p = \{(\tilde{x}_1, \tilde{y}_1), \dots, (\tilde{x}_{\tilde{n}}, \tilde{y}_{\tilde{n}})\} \subseteq \mathcal{F}$, then there exists a set of at most \tilde{n} poisoned points $\tilde{\mathcal{D}}_p$ (possibly with fractional copies) that also attains $\hat{\theta}$ but only contains 2 distinct points, one from each class.

More generally, the above statement is true for any margin-based model with loss of the form $\ell(\theta; x, y) = c(-y\theta^{\top}x)$, where $c : \mathbb{R} \to \mathbb{R}$ is a convex, monotone increasing, and twice-differentiable function, and the ratio of second to first derivatives c''/c' is monotone non-increasing.

Proof From Proposition 7 (SVMs), Proposition 8 (margin-based losses), and Corollary 9 (logistic regression), we have that the Carathéodory number of $\mathcal{G}_{\hat{\theta}}$ (the set of scaled possible gradients) for each of these models is at most 2, regardless of $\hat{\theta}$. By Proposition 4, we conclude that only 2 distinct points are necessary. In particular, since $\mathcal{G}_{\hat{\theta}}$ can be represented in each of these cases as the union of two convex sets, one for each class, we need 1 distinct point from each class to realize any data poisoning attack.

The general approach of finding the Carathéodory number of the set of scaled possible gradient can be applied to other models beyond those that we consider in this paper. As one example, we can extend the above approach to the setting of a multi-class SVM:

Proposition 10 (k(k-1) distinct points suffice for a k-class SVM) Consider the setting of Proposition 7, but with a k-class SVM.

If a parameter $\hat{\theta}$ is attainable by any set of \tilde{n} poisoned points $\mathcal{D}_p = \{(\tilde{x}_1, \tilde{y}_1), \dots, (\tilde{x}_{\tilde{n}}, \tilde{y}_{\tilde{n}})\} \subseteq \mathcal{F}$, then there exists a set of at most \tilde{n} poisoned points $\tilde{\mathcal{D}}_p$ that also attains $\hat{\theta}$, but that only has k-1 distinct values of \tilde{x} for each distinct \tilde{y} , for a total of k(k-1) distinct points.

Proof We follow the multi-class SVM formulation described in Crammer and Singer (2002), which has parameters $\theta = [\theta_1; \theta_2; \dots; \theta_k] \in \mathbb{R}^{kd}$ for each class $y = 1, 2, \dots, k$, where d is the dimensionality of the feature space \mathcal{X} . The loss on an individual point is given by

$$\ell(\theta; x, y) = \max(0, 1 - \theta_y^\top x + \max_{i \neq y} \theta_i^\top x). \tag{34}$$

This reduces to the above formulation for a binary (2-class) SVM by setting $\theta_{-1} = -\theta_{+1}$. Let $i_{x,y} = \arg\max_{i \neq y} \theta_i^{\top} x$, and let $x^{(i)} = [\underbrace{0, \dots, 0}_{(i-1)d \text{ zeroes}}, x, \underbrace{0, \dots, 0}_{(n-i)d \text{ zeroes}}]$. Without loss of

generality, we can ignore subgradients and take $\nabla_{\theta} \ell(\theta; x, y) = -x^{(y)} + x^{(i_{x,y})}$, following a similar argument to that used in Proposition 7.

Define $\mathcal{G}_{i,j} = \{-\alpha x^{(i)} + \alpha x^{(j)} : 0 \le \alpha \le 1, (x,i) \in \mathcal{F}\}$. We have that $\mathcal{G}_{\hat{\theta}} = \bigcup_{i=1}^k \bigcup_{j \ne i}^k \mathcal{G}_{i,j}$, and since each $\mathcal{G}_{i,j}$ is a convex set, by Lemma 6, the Carathéodory number of \mathcal{G} is at most k(k-1).

Appendix B. Label Flip Attacks

Label flip attacks are a popular strategy in the literature for executing data poisoning attacks (Biggio et al., 2011; Xiao et al., 2012, 2015). In a label flip attack, the attacker is given a set of real data $\mathcal{D}_{\text{pool}} = \{(\tilde{x}_i, \tilde{y}_i)\}_{i=1}^{n_{\text{pool}}}$. To form the poisoned data \mathcal{D}_{p} , the attacker chooses ϵn points (possibly repeated) from $\mathcal{D}_{\text{pool}}$ and flips their labels.

Many ways of choosing which points to flip have been proposed (see Xiao et al. (2015) for a review). Here, we consider the **alfa** attack from Xiao et al. (2012). **alfa** seeks to add points that have a high loss under the original (clean) model $\theta^* \stackrel{\text{def}}{=} \operatorname{argmin}_{\theta} L(\theta; \mathcal{D}_c)$ but a low loss under the final (poisoned) model $\hat{\theta} \stackrel{\text{def}}{=} \operatorname{argmin}_{\theta} L(\theta; \mathcal{D}_c \cup \mathcal{D}_p)$. Concretely, it solves:

$$\begin{aligned} \underset{\mathcal{D}_{p}}{\text{maximize}} \quad & L(\hat{\theta}; \mathcal{D}_{p}) - L(\theta^{*}; \mathcal{D}_{p}) \\ \text{s.t.} \quad & |\mathcal{D}_{p}| = \epsilon |\mathcal{D}_{c}| \\ & (\tilde{x}, -\tilde{y}) \in \mathcal{D}_{pool} \quad \forall (\tilde{x}, \tilde{y}) \in \mathcal{D}_{p} \\ & \hat{\theta} = \underset{\theta}{\text{argmin}} L(\theta; \mathcal{D}_{c} \cup \mathcal{D}_{p}) \end{aligned}$$

We adapt the original alfa attack to our setting in the following two ways:

- 1. We constrain all poisoned points in \mathcal{D}_p to lie in the feasible set \mathcal{F}_{β} .
- 2. We set $\mathcal{D}_{pool} = \mathcal{D}_{test}$; that is, the attacker gets to add points from the flipped test set. Intuitively, adding flipped versions of test points to the training set should cause the model to wrongly classify those test points, in line with the attacker's goal of increasing the model's loss on the test set.

We evaluated this variant of **alfa** on the Enron dataset. To make it easier for the attacker, we only considered the **L2** defense. Our implementation of the **alfa** attack was not able to increase the test error much, achieving a 2% increase in test error with $\epsilon = 3\%$. In contrast, the attacks we introduce in the main text achieve an increase in test error of 15% to 20%, despite having to deal with more sophisticated defenses. Our conclusion is that only modifying the labels \tilde{y} , as in label flip attacks, does not give the attacker much power relative to being able to modify \tilde{x} as well.

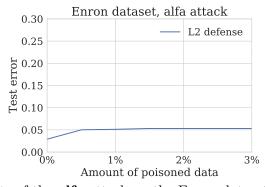


Figure 12: Results of the **alfa** attack on the Enron dataset and **L2** defense.