An Exploration of Poisoning Attacks on Data-based Decision Making

Sarah Eve Kinsey, Wong Wai Tuck, Arunesh Sinha, and Thanh H. Nguyen

[1] University of Oregon arbutler@cs.uoregon.edu
[2] Singapore Management University wt.wong.2020@msc.smu.edu.sg
[3] Rutgers University arunesh.sinha@rutgers.edu
[4] University of Oregon thanhhng@cs.uoregon.edu

Abstract. Many real-world problems involve building a predictive model about an adversary and then determining a decision accordingly, including two-stage (predict then optimize) and decision focused (joint predict and optimize) approaches. The involvement of a predictive model learned from adversary's behavior data poses a critical threat that an adversary can influence the learning process, which will ultimately deteriorate the end-goal decision quality. In this paper, we study the problem of poisoning attacks in this data-based decision making setting. That is, the adversary can alter the training data by injecting some perturbation into the data to a certain limit that can substantially change the final decision outcome in the end towards the adversary goal. To our knowledge, this is the first work that studies poisoning attacks in such data-based decision making scenarios. In particular, we provide the following main contributions. We introduce a new meta-gradient based poisoning attack for various types of predict and optimize frameworks. We compare to a technique shown effective in computer vision. We find that the complexity of the problem makes attacking decision focused model difficult. We show that an attack crafted against a two-stage model is effectively transferable to a decision-focused model.

1 Introduction

As machine learning has been gaining a lot of adventions and interests from both research and industrial communities, the opportunities for and the potential cost of failure grow. Some sources of failure are well explored, such as poorly chosen models and biased datasets. More recently, research has also considered another avenue for failure: intelligent adversaries that wish to manipulate the results of machine learning models [18, 10]. For example, adversaries can perform *evasion attacks* [5, 16, 23] to alter the classification of particular samples at test time; this requires access to some data that will be taken as input by a pre-trained model. Alternatively, with access to the training data, attackers can perform *poisoning attacks* [13, 6, 27]. The goal of poisoning attack is to manipulate the training data such that the resulting model offers advantage to the adversary. *Adversarial machine learning* is the field that includes study of both evasion and poisoning attacks, as well as design of models resistant to these attacks.

Another emerging area of study is that of *data-based decision making*. Many machine learning applications involve a data to decision pipeline: first using known data construct a predictive model, then apply the predictive model to unknown data, and lastly make decisions based on those predictions. Traditional approaches here have been *two-stage*, with the predictive model being optimized solely for its prediction accuracy [29, 8, 20, 34, 30]. If the prediction model is perfect over the whole prediction space, the two-stage approach would be optimal. However, complicated prediction boundaries in high dimension spaces can never be modeled perfectly even with large but finite data; in fact, for data driven decision making the end goal is to make the best decisions possible, but the prediction model itself is not being optimized with that goal in mind. As a consequence of this observation, a method often referred to as *decision focused* learning seeks to directly integrate the decision optimizer into the prediction model during training. Hence, decision focused learning uses the decision quality to train the network. Updating the model via gradient descent, then, can be accomplished by differentiating *through* the solution to the decision optimization. This approach has proven more effective than corresponding two-stage models in some applications. However, this approach is significantly more computationally expensive, as each forward pass in the training process requires solving the optimization.

Our work lies at the intersection of data-based decision making and adversarial learning. We investigate the vulnerabilities of data-based decision making methods by developing poisoning attacks against these methods. To our knowledge, our work is the first one exploring this topic. Specifically, we look into using *end-to-end attacks* against both the aforementioned data-based decision methods designed for convex optimization, as well as a third model which we call the *simple joint* model. Here, the optimizer is itself approximated by a neural network. Furthermore, as it is important to understand the *transferability* of poisoning attacks between different models, we also investigate how effectively our generated attacks can be transferred beyond the originally targeted method (e.g. computing an attack against a two stage model and then also testing the generated poison on a decision focused model).

Our *first* contribution is to create a meta-gradient based poisoning attack. Put simply, we unroll the target model's training procedure (which consists entirely of differentiable steps) to differentiate through the training and calculate gradients of the attacker's loss function with respect to the training data itself. Then, we use these gradients to perform projected (into the feasible space defined by constraints on the attack) gradient descent.

Our *second* contribution is to demonstrate that existing state of the art methods in other domains (specifically Metapoison [12] in computer vision) may not be directly applicable to the field of data-based decision making. We accomplish this by attacking a simple data-based decision making learner (using Metapoison to solve the attack) as well as testing the found attack on both two-stage and decision focused learners. The ineffectiveness of this approach for our problem suggests that new techniques may have to be developed for poisoning attacks on data-based decision making models.

Our experiments yield several findings that should be of use to future research. Most notably, attacking a decision-focused learner directly is a particularly difficult ask due to the complexity of the learner's training process. Beyond the (significant) computational requirements of solving the attack, any stability or precision issues within the learner's gradient calculation are compounded when computing the meta-gradient. Common machine learning pitfalls such as exploding or vanishing gradients appear frequently and are harder to counteract. Furthermore, the complexity of the solution space (which scales with model size, optimization objective, and constraints) means that many optima of various quality exist, and finding a good one with gradient descent is not guaranteed. These effects are less noticeable when attacking the two-stage model or the simple joint model, as their training gradient updates do not involve backpropagating through an optimization problem.

Furthermore, we investigate the transferability of our method's attacks. Previous work has shown the transferability of meta-gradient based poisoning attacks [21]. Our experiments show that this property still applies, to varying degrees, across data-based decision making methods. This finding aligns with the general *transferability phenomenon* found in adversarial machine learning [22]. Primarily, we observe that poisons created against a two-stage learner effectively transfer to a decision-focused learner.

2 Related Work

Decision Focused Learning. Decision-focused learning is an approach that has been applied to discrete [31] and convex [32] optimization, as well as non-convex optimization in security games [24]. The key idea here is that, instead of a traditional two-stage "predict then optimize" approach to solving problems, the decision maker instead uses the decision quality itself as the training loss. This is done by backpropagating *through* the optimization problem, which can be accomplished by applying the implicit function theorem to the `KKT` conditions of that optimization [31]. Our proposed attack could be called a decision-focused attack; we differentiate through the entire data-based decision making problem to optimize our poison.

Adversarial Learning. Adversarial learning is a subfield of machine learning focused on attacking models. The attack formulated in this work is analogous to a *causative attack* (or poisoning attack) in adversarial learning [11, 17, 33, 35, 12]. A significant difference between our work and typical adversarial learning is that our attacker has end goals beyond minimizing prediction accuracy. While our attack does target the training process, the attacker's objective is to manipulate the decision outcome of the optimization problem.

Meta Learning. In machine learning, meta learning is an approach designed to optimize the training process itself. Historically, meta learning was focused on *improving* models, though more recently, meta learning based attacks have

proven effective. Muñoz-González et al. used a metagradient method to optimize a poisoning attack by back-propagating *through* the learning process [21]. They demonstrated that this approach was effective against a variety of decision makers, for multiple different tasks. Interestingly, they found that these poisoning attacks could be effectively transferred to models other than the one against which they were optimized [21]. Zugner et al. utilized the metagradient method to attack graph learning problems, creating an attack capable of dramatically reducing global node classification accuracy [35]. MetaPoison uses shallow metagradients averaged over multiple models at each poison optimization step to produce a robust yet subtle attack on image classification that can be effectively transferred beyond the original target model [12]. Similar to these papers, our work focuses on a metagradient poisoning attack. Our contribution is in extending metagradient attacks to data based decision making models, and providing a detailed overview of the challenges in creating poisoning attacks in this setting.

3 Data-based Decision Making

Data-based decision making refers to a common paradigm in artificial intelligence in which we are concerned with three related pieces of information: directly observable data (denoted by u), data that will be unobservable at test time (denoted by θ), and a *decision* that must be made (denoted by x). The decision, x, depends directly on θ, which in turn can be predicted based on u. The ultimate goal in a data-based decision making problem is to find an optimal decision to maximize a utility function, abstractly represented as follows:

$$\max_{x \in X} f(x, \theta)$$

where x is the decision variable and $X \subseteq \mathbb{R}^K$ is the set of all feasible decisions. Note that the objective, f, depends directly on the *unobservable* parameter θ, which must be inferred from the correlated observable data, u. In this work, we focus on the problem setting in which the decision space X can be represented as a set of linear constraints $X = \{x \in \mathbb{R}^K : Ax \leq b\}$ where (A, b) are constant.

In literature, there are two main approaches used to tackle the data-based decision making problem. The first approach, named **two-stage approach**, divides the problem into two separate phases. The first phase is the learning phase in which the unobserved parameter θ is learnt based on some training dataset $\mathcal{D} = \{(u_1, \theta_1), (u_2, \theta_2), \cdots, (u_n, \theta_n)\}$ in which each data point i is associated with a feature vector $u_i \in \mathbb{R}^d$ and a true label $\theta_i \in \mathbb{R}^K$ ($\theta_i \in \mathbb{N}^K$ if it is categorical). Then in the second phase which is called the decision-making phase, the decision x will be optimized based on the learning outcome θ. The second approach, named **decision-focused** learning, on the other hand, considers a single end-to-end pipeline (with an intermediate learning layer) that attempts to directly optimize the decision based on the training data $\mathcal{D}$. In addition to these two main approaches, in this paper, we create a third simple approach, named **simple joint approach** that formulates the data-based decision making

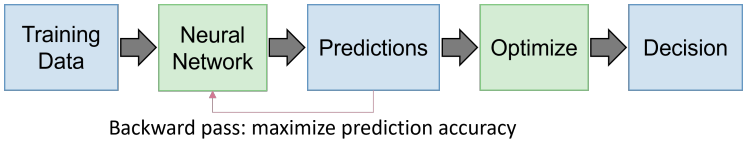


Fig. 1: Depiction of a two-stage learner

as a simple learning problem. We consider this approach as a baseline to study poisoning attacks in this data-based decision making setting.

In the following, we first describe in details all these three approaches. We then present our optimization formulations to compute poisoning attacks to these three approaches, which are challenging to solve. Our proposed methodology to solve these optimization problems will be presented in Section 4.

3.1 Two-Stage Approach

Learner Description The traditional approach to data-based decision making is *two-stage* [29, 8, 30]. The first of these stages is predicting the unknown parameter θ from the observed feature vector u. The second stage, then, is to compute the optimal x given the predicted θ (Figure 1). Predicting the *unknown* parameter θ can be done using a parametric model, denoted by $\hat{\theta} = g(u, w)$. Here, w is the model parameter that needs to be determined. Given a training dataset $\mathcal{D} = \{(u_1, \theta_1), (u_2, \theta_2), \cdots, (u_n, \theta_n)\}$ in which each data point i is associated with a feature vector $u_i \in \mathbb{R}^d$ and a true label $\theta_i \in \mathbb{R}^K$ ($\theta_i \in \mathbb{N}^K$ if it is categorical), the decision maker first trains a predictive model $g(u, w)$ to predict the label of a data point u. The learner seeks an optimal model parameter w^* that minimizes the training loss, abstractly formulated as follows:

$$\min_w \mathcal{L}(\mathcal{D}, w)$$

For example, one can use mean squared error as the training loss:

$$\mathcal{L}(\mathcal{D}, w) = \frac{1}{n} \sum\nolimits_i (\theta_i - g(u_i, w))^2$$

Once the model has been trained (yielding w^*), the decision maker can use observed u values to predict θ value (i.e., $g(u, w^*)$), then use that prediction to find an optimal decision by solving the following optimization problem:

$$\max_{x \in X} f(x, g(u, w^*))$$

Poisoning Attack Formulation In designing poisoning attacks, we assume an adversary can alter the training data by injecting a small perturbation to every data point. More specifically, each feature vector u_i can be altered by adding a small quantity ϵ_i with the constraint that $lb_i \leq \epsilon_i \leq ub_i$. Here, $lb_i < 0$ and

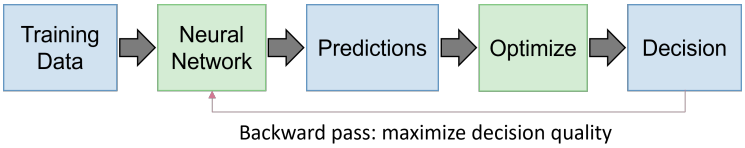


Fig. 2: Depiction of a decision focused learner

$ub_i > 0$ represent the maximum perturbation the adversary can apply to the data point i. Intuitively, (lb_i, ub_i) captures the adversary's capability. The adversary attempts to optimize some goal, for example, minimizing the decision maker's utility in the test set or forcing the decision maker to produce a particular target decision output for some data points in the test set. We represent this poisoning attack on a two-stage model with the following general formulation:

$$\min \mathcal{L}^{adv}(x^*, \theta^{target}) \tag{1}$$
$$\text{s.t. } x^* \in \text{argmax}_{x \in X} f(x, g(u^{target}, w^*)) \tag{2}$$
$$w^* \in \text{argmin}_w \mathcal{L}(\mathcal{D}^{poison}, w) \tag{3}$$
$$\mathcal{D}^{poison} = \{(u_1 + \epsilon_1, \theta_1), \cdots, (u_n + \epsilon_n, \theta_n)\} \tag{4}$$
$$\epsilon_i \in [lb_i, ub_i], \forall i = 1, 2, \cdots, n \tag{5}$$

where $(u^{target}, \theta^{target})$ is the adversary's target element. Line 1 simply represents a general objective for the adversary. For example, if the adversary's goal is to minimize the decision maker's utility on this target, then $\mathcal{L}^{adv}(x^*, \theta^{target}) = f(x^*, \theta^{target})$. Line 2 is the optimization problem solved by the learner given the network output. Equation 3 then represents the optimal network weights as a function of the learner's training. Next, line 4 denotes the training dataset, altered by the attacker with poison values ϵ. Lastly, line 5 denotes the restrictions on the attacker's power, specifically a magnitude constraint on each poison element.[5] Solving the above optimization problem optimally is challenging since it has multiple connected levels of optimizations.

3.2 Decision Focused Approach

Learner Description While the two-stage approach is straightforward and effective, its training process is disconnected from the end goal of the system. More specifically, the model is being trained for prediction accuracy, whereas the ultimate objective is to produce good decisions [32].

[5] Note that our formulation can be generalized to multiple targeted data points in the test set by taking the sum of losses over these data points. In addition, this can be also extended to incorporate perturbations on labels θ_i by introducing additional perturbation variables α_i to add to the labels.

A recent approach called *decision focused* learning seeks to bridge the disconnect between the training and the decisions produced, while still utilizing an explicit optimization solver (Figure 2). In theory, this approach can improve final decision quality by concentrating the (inevitable) prediction errors in areas that will have the least detrimental effect. For each training data point, θ is predicted from u and the optimization problem is solved to produce x. Then, the network weights are updated via gradient descent to maximize the decision quality. Intuitively, we can think of this as incorporating a convex optimization layer as the last layer of a neural network. This method can give improved results over the two-stage approach, at the cost of training time [32]. Essentially, in a decision-focused approach, the loss function the learner minimizes is the negative mean decision quality:

$$\min_{w} \ \mathcal{L}(\mathcal{D}, w)$$
$$\text{where } \mathcal{L}(\mathcal{D}, w) = -\frac{1}{n}\sum\nolimits_i f\Big(\theta_i, x^*(\hat{\theta}_i)\Big)$$

In this case, x^* is a result of solving the following optimization problem:

$$x^*(\hat{\theta}_i) \in \text{argmax}_{x \in X} f(x, \hat{\theta}_i)$$
$$\text{where } \hat{\theta}_i = g(u_i, w) \text{ is the network output.}$$

Unlike the two-stage approach, here one must differentiate *through* the decision optimization problem to optimize the model parameters w. This can be accomplished by using the implicit function theorem on the KKT conditions of the optimization problem [2].

Poisoning Attack Formulation Given the decision-focused formulation, we now can represent the problem of finding an optimal poisoning attack as the following optimization problem:

$$\min \mathcal{L}^{adv}(x^*, \theta^{target}) \tag{6}$$
$$\text{s.t. } x^* \in \text{argmax}_{x \in X} f(x, g(u^{target}, w^*)) \tag{7}$$
$$w^* \in \text{argmin}_w \left[\mathcal{L}(\mathcal{D}^{poison}, w) = -\frac{1}{n}\sum\nolimits_i f\Big(\theta_i, x^*(\hat{\theta}_i)\Big)\right] \tag{8}$$
$$\text{given } x^*(\hat{\theta}_i) \in \text{argmax}_{x \in X} f(x, \hat{\theta}_i) \tag{9}$$
$$\text{and } \hat{\theta}_i = g(u_i + \epsilon_i, w) \text{ is the network output.} \tag{10}$$
$$\mathcal{D}^{poison} = \{(u_1 + \epsilon_1, \theta_1), \cdots, (u_n + \epsilon_n, \theta_n)\} \tag{11}$$
$$\epsilon_i \in [lb_i, ub_i], \forall i = 1, 2, \cdots, n \tag{12}$$

At a high level, the general attack formulation in this setting is similar to the two-stage case. However, solving the above optimization problem is much more challenging since the learner's training Eq. (8 – 10) involves an inner optimization layer which depends on the decision optimizer for every training data point.

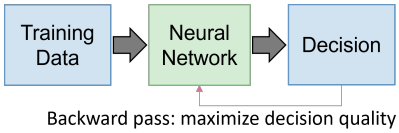


Fig. 3: Depiction of a learner using the simple joint approach

3.3 Simple Joint Approach

Learner Description A naive approach to data based decision making is to train a parametric model using the features (u) to directly predict the optimal decision (x) (Figure 3). Similar to the decision focused approach, we use negative mean decision quality as the loss function:

$$\min_w \ \mathcal{L}(\mathcal{D}, w)$$
$$\mathcal{L}(\mathcal{D}, w) = -\frac{1}{n}\sum\nolimits_i f\Big(\theta_i, \hat{x}(u_i)\Big)$$

In this case, however, $\hat{x}$ itself is the network output: $\hat{x}(u_i) = g(u_i, w)$. This bypasses the need to directly predict the labels ($\hat{\theta}$). Intuitively, we can think of this as implicitly learning the predictive task "inside" of the network.

Alternatively, we could solve the optimization problem for each training set instance prior to training (producing x_i^* where $x_i^* \in \text{argmax}_{x \in X} f(x, \theta_i)$) and then train the network to produce decisions as close as possible to these x^* values. In this method, we could use MSE as the loss function: $\mathcal{L}(\mathcal{D}, w) = \frac{1}{n}\sum_{i=1}^{n}(x_i^* - g(u_i, w))^2$. However, we found this approach less effective and more prone to overfitting than directly maximizing decision quality.

One complication when training a network to solve optimization problems is the constraints (if any exist) on the solution. Inspired by Shah et. al [28], we utilize a specially designed neural network layer to enforce constraints throughout the training process, ensuring valid decisions are made [28].

In practice, this naive approach is often ineffective as training networks to directly solve optimization problems is difficult. However, we investigate this simple model as a target for generating poisoning attacks that can then be transferred to the more sophisticated models.

Poisoning Attack Formulation The attack formulation here is similar to the previous cases. The difference is in the second line; rather than producing predictions, this simple joint model simply treats the network output as the

decision itself, and is trained accordingly:

$$\begin{aligned}
&\min \mathcal{L}^{adv}(x^*, \theta^{target}) \\
&\text{s.t. } x^* = g(u^{target}, w^*) \\
&\quad w^* \in \text{argmin}_w \left[\mathcal{L}(\mathcal{D}^{poison}, w) = -\frac{1}{n} \sum_i f\Big(\theta_i, \hat{x}(u_i + \epsilon_i)\Big) \right] \\
&\quad \mathcal{D}^{poison} = \{(u_1 + \epsilon_1, \theta_1), \cdots, (u_n + \epsilon_n, \theta_n)\} \\
&\quad \epsilon_i \in [lb_i, ub_i], \forall i = 1, 2, \cdots, n
\end{aligned}$$

4 Attack Generation Methodology

To solve the aforementioned optimization problems and determine poisoning attacks against each of these decision making approaches, we follow projected gradient descent. The core of gradient descent is to compute the gradient of the adversary loss $\mathcal{L}^{adv}$ with respect to the data perturbation ϵ, denoted by $\frac{d\mathcal{L}^{adv}}{d\epsilon}$. This gradient computation is challenging given that all the optimization problems involve multiple connected optimization levels.

Despite the differences among the aforementioned three data-based decision approaches, we employed two main computation techniques: (i) *computing gradients via meta gradient* [4]— the main idea of this technique is to differentiate through the gradient descent steps in solving inner optimization levels. The main advantage of this technique is that it can be can be applied for any non-convex optimization problems. One disadvantage of this technique is that it is generally computationally expensive; and (ii) *computing gradient via implicit function theorem* [7, 31] — the main idea of this technique is to leverage convexity property, allowing us to differentiate through the KKT optimality condition. This technique is significantly less computationally expensive compared to the first technique. However, this technique is only applicable for convex optimization problem. Therefore, depending on the convexity of the problems, we then decide on one of these two techniques.

In the following, we first present in detail our proposed method to compute attacks to two-stage learning. Later, we will mainly highlight the differences or challenges regarding the decision-focused learning and the simple joint learning.

4.1 Attack to Two-Stage Approach

To solve the poisoning attack in this setting using gradient descent, the key is the gradient calculation of $\frac{d\mathcal{L}^{adv}}{d\epsilon}$, which can be decomposed into different gradient components via the chain rule:

$$\frac{d\mathcal{L}^{adv}}{d\epsilon} = \frac{d\mathcal{L}^{adv}}{dx^*} \frac{dx^*}{dg} \frac{dg}{d\epsilon} \qquad \frac{dg}{d\epsilon} = \frac{dg}{dw^*} \frac{dw^*}{d\epsilon}$$

Computing $\frac{d\mathcal{L}^{adv}}{dx^*}$ is straightforward, and $\frac{dg}{dw^*}$ is a result of the standard neural network back-propagation computation. On the other hand, computing the gradient components, $\frac{dx^*}{dg}$, the gradient of the optimal decision with respect to the label prediction $g(u^{target}, w^*)$, and $\frac{dw^*}{d\epsilon}$, the gradient of the optimal model parameter w^* w.r.t the perturbation ϵ, is not straightforward. This is because there is no explicit close-formed representation of x^* and w^* as a function of g and ϵ respectively, despite the fact that x^* depends on g and w^* depends on ϵ. In the following, we present our meta-gradient based method to approximate $\frac{dw^*}{d\epsilon}$ given the learning part (neural network function) is non-convex. We will then present the implicit function theorem based method to compute $\frac{dx^*}{dg}$ since the decision optimization part is convex.

Computing decision gradient via implicit function theorem We focus on the problem setting in which the decision optimization is convex (i.e., the utility function $f(x, \theta)$ is convex in the decision variable x). This convexity setting has been widely considered in previous studies on data-based decision making [32, 31, 7, 1]. Based on this convexity characteristic, we leverage the implicit function theorem [15] to differentiate through the decision-optimization layer (i.e., computing $\frac{dx}{dg}$). Given the predicted value $\hat{\theta} = g(u^{target}, w^*)$, the decision-optimization component is formulated as a convex optimization problem:

$$\max_x \ f(x, \hat{\theta}) \ \text{ s.t. } \ Ax \leq b$$

Since this is a convex optimization problem, any solution that satisfies the following KKT conditions is optimal:

$$\begin{aligned} &-\nabla_x f(x, \hat{\theta}) + \lambda \cdot \nabla_x (Ax - b) = 0 \\ &\lambda \cdot (Ax - b) = 0 \\ &Ax \leq b, \lambda \geq 0 \end{aligned}$$

where λ is the dual variable. Observe that the first equation indicates that x and λ are functions of $\hat{\theta}$. Based on the implicit function theorem, we can differentiate through the first two equations to obtain the following gradient computation:

$$\begin{bmatrix} \frac{dx}{d\hat{\theta}} \\ \frac{d\lambda}{d\hat{\theta}} \end{bmatrix} = \begin{bmatrix} \nabla_x^2 f(x, \hat{\theta}) & A^T \\ diag(\lambda)A & diag(Ax - b) \end{bmatrix}^{-1} \begin{bmatrix} \frac{d\nabla_x f(x, \hat{\theta})}{d\hat{\theta}} \\ 0 \end{bmatrix} \tag{13}$$

Computing learning gradient via meta gradient While we can leverage the convexity of decision optimization to compute the gradient of a decision with respect to the coefficients (as will be done when attacking the more complex models), we cannot apply the same approach for computing the learning gradient, $\frac{dw^*}{d\epsilon}$. This is because model learning is generally a non-convex optimization problem (as neural network models are non-convex in general). On the other hand, the implicit function theorem approach is most usefully applied to convex optimization. In order to tackle this challenge, we adopt the meta-gradient method

[3].[6] This method works by assuming the model learning problem is solved via gradient descent. This is a reasonable assumption since neural network training typically relies on gradient descent method and its variants.

Based on this assumption, we can differentiate through the gradient descent steps. More specifically, we're concerned with the model's learning problem, abstractedly represented as follows:

$$\min_{w} \mathcal{L}(\mathcal{D}^{poison}, w)$$
$$\text{where } \mathcal{D}^{poison} = \{(u_1 + \epsilon_1, \theta_1), \cdots, (u_n + \epsilon_n, \theta_n)\}$$

At each gradient step t, given the previous value of the model parameters w_{t-1}, the gradient descent update is as follows: $w_t = w_{t-1} - \delta \frac{d\mathcal{L}}{dw_{t-1}}$, where δ is the learning rate. Note that $\mathcal{L}$ is a function of the perturbation variables $\epsilon = \{\epsilon_1, \cdots, \epsilon_n\}$. Therefore, w_t is also a function of ϵ (except for w_0 which is the initial value, a constant). As a result, we can differentiate through this gradient step as follows:

$$\frac{dw_t}{d\epsilon} = \frac{dw_{t-1}}{d\epsilon} - \delta \frac{dG}{d\epsilon}$$

where $G(w_{t-1}, \epsilon) = \frac{d\mathcal{L}}{dw_{t-1}}$. By applying the chain rule, we obtain:

$$\frac{dG}{d\epsilon} = \frac{\partial G}{\partial \epsilon} + \frac{\partial G}{\partial w_{t-1}} \cdot \frac{dw_{t-1}}{d\epsilon}$$

If we run gradient descent in T steps, we can approximate the gradient of the optimal w^* with respect to perturbations ϵ as follows: $\frac{dw^*}{d\epsilon} \approx \frac{dw_T}{d\epsilon}$.

Projected gradient descent algorithm Given this gradient computation, we illustrate our approach in Algorithm 1 where we run an iterative projected gradient descent process to compute an optimal attack. At each iteration j, given the current value of perturbation variables ϵ, Algorithm 1 first runs another inner gradient descent process to optimize the parameters w of the predictive model $g(u, w)$ based on the poison data $\mathcal{D}^{poison}$ (lines 5-13). At the end of this inner process, we obtain a trained model w^*. During this process, we simultaneously compute the learning gradient $\frac{dw^*}{d\epsilon}$.

Given the trained model w^*, Algorithm 1 proceeds into the decision optimization to compute the optimal decision x^* w.r.t the target u^{target} (line 14). Along with that computation, the gradient $\frac{dx^*}{dg}$ is computed (line 15). Finally, we update the value of ϵ based on the previous gradient computation (lines 16-17). This entire procedure (lines 5-17) is repeated until we reach a local optimal value of ϵ or reach the predetermined maximum number of iterations $nIter$.

[6] We can also apply this method to compute the decision gradient. However, meta-gradient is much more computationally expensive compared to the implicit function theorem method for convex problems.

Algorithm 1: Poisoning Attack Generation for Two-Stage Learning

Input: training data $\mathcal{D} = \{(u_1, \theta_1), (u_2, \theta_2), \cdots, (u_n, \theta_n)\}$;
Input: target $(u^{target}, \theta^{target})$;
Randomly initialize perturbation values $\epsilon = \{\epsilon_1, \cdots, \epsilon_n\}$.
for $j = 1 \rightarrow nIter$ **do**
    `// Model learning`
    Initialize optimal learning loss $optL = \infty$;
    for $r = 1 \rightarrow nRound$ **do**
        Randomly initialize model parameter values w_0;
        for $t = 1 \rightarrow T$ **do**
            Update $w_t = w_{t-1} - \delta \frac{d\mathcal{L}}{dw_{t-1}}$;
            Differentiate $\frac{dw_t}{d\epsilon} = \frac{dw_{t-1}}{d\epsilon} - \delta \frac{d\left(\frac{d\mathcal{L}}{dw_{t-1}}\right)}{d\epsilon}$;
        if $\mathcal{L}(\mathcal{D}^{poison}, w_T) < optL$ **then**
            Update optimal learning $w^* = w_T$;
            Update learning gradient: $\frac{dw^*}{d\epsilon} = \frac{dw_T}{d\epsilon}$;
    `// Decision optimizing`
    Compute optimal decision based on the learnt model w^*: $x^* \in \text{argmax}_{x \in X} f(x, g(u^{target}, w^*))$
    Compute decision gradient w.r.t $\hat{\theta} = g(u^{target}, w^*)$ using Eq. (13)
    `// Projected gradient step`
    Update perturbation variable $\epsilon = \epsilon - \delta \frac{d\mathcal{L}^{adv}}{d\epsilon}$ where: $\frac{d\mathcal{L}^{adv}}{d\epsilon} = \frac{d\mathcal{L}^{adv}}{dx^*} \frac{dx^*}{dg} \frac{dg}{dw^*} \frac{dw^*}{d\epsilon}$;
    Project ϵ_i to feasible perturbation space: $[lb_i, ub_i]$ $\forall i$;
return ϵ;

4.2 Attack to Decision Focused Approach

Similar to the two-stage approach, in this setting, we aim to compute the gradient $\frac{d\mathcal{L}^{adv}}{d\epsilon}$, which can be decomposed into multiple components using chain rule:

$$\frac{d\mathcal{L}^{adv}}{d\epsilon} = \frac{d\mathcal{L}^{adv}}{dx^*} \frac{dx^*}{dg} \frac{dg}{d\epsilon} \qquad \frac{dg}{d\epsilon} = \frac{dg}{dw^*} \frac{dw^*}{d\epsilon}$$

However, as the two methods use different training processes, the details of the learning gradient calculation ($\frac{dw^*}{d\epsilon}$) differ significantly. In fact, it becomes much more complicated and computationally expensive due to the involvement of the optimizer in the training process itself.

Indeed, recall that for the calculation of the learning gradient ($\frac{dw^*}{d\epsilon}$), in general, we follow gradient descent at every to solve the model learning problem and then differentiate through the gradient steps as explained in the previous section. That is, we have the following differentiation update:

$$\frac{dw_t}{d\epsilon} = \frac{dw_{t-1}}{d\epsilon} - \delta \frac{d\left(\frac{d\mathcal{L}}{dw_{t-1}}\right)}{d\epsilon}$$

where $\mathcal{L}$ is the training loss. In two-stage approach, this training loss has a closed-formed representation as a function of ϵ. Therefore, the above gradient computation is straightforward. On the other hand, in decision-focused approach, the model training is represented in Eq. (8–10), in which multiple decision optimizations for every data point is involved. The gradient $\frac{d\mathcal{L}(\mathcal{D}^{poison},w)}{dw}$ now depends on the gradient of the optimal decision w.r.t the prediction outcomes $\frac{dx^*(\hat{\theta}_i)}{d\hat{\theta}_i}$ for all i, since we have according to the chain rule:

$$\frac{d\mathcal{L}}{dw} = \sum_i \frac{d\mathcal{L}}{dx^*(\hat{\theta}_i)} \frac{dx^*(\hat{\theta}_i)}{d\hat{\theta}_i} \frac{d\hat{\theta}_i}{dw}$$

Computing the gradient $\frac{dx^*(\hat{\theta}_i)}{d\hat{\theta}_i}$ for all data points can be done via implicit function theorem as discussed in Section 4.1, which already involves complex computations including inverse matrix computation and the second derivative computation, etc. As a result, it becomes very challenging to take a further gradient step of $\frac{d\left(\frac{d\mathcal{L}}{dw}\right)}{d\epsilon}$. We discuss this challenge in the experiment section.

4.3 Attack to Joint Simple Approach

Finally, solving the attack on the joint sample approach is the simplest:

$$\frac{d\mathcal{L}^{adv}}{d\epsilon} = \frac{d\mathcal{L}^{adv}}{dx^*} \frac{dx^*}{d\epsilon} \qquad \frac{dx^*}{d\epsilon} = \frac{dx^*}{dw^*} \frac{dw^*}{d\epsilon}$$

In this case, x^* is simply the output of the neural network. Computing $\frac{d\mathcal{L}^{adv}}{dx^*}$ is straightforward, and $\frac{dx^*}{dw^*}$ is a result of the standard neural network back-propagation computation. The only challenging component here is $\frac{dw^*}{d\epsilon}$ as w^* is a function of ϵ yet cannot be expressed in closed form. As when attacking the other models, we use the metagradient method to calculate this.

5 Experiment Setup

5.1 Attack Methods

For our experiments, we utilize three different methods to generate attacks and compare their effectiveness. Starting with the simplest model, the first method is based on MetaPoison [12] and is formulated against the naive end-to-end learner. More specifically, this attack utilizes multiple target models (each at a different stage of training) and *averages* their metagradients (limited to 2 training steps). Then, the attack is optimized alongside the target models. This method was found to produce effective, unnoticeable, and transferable attacks in the computer vision domain [12]. Our idea is to use this method against a simple learner in the data-based decision making domain to produce attacks

that can then be leveraged against the more sophisticated learners (two-stage and decision-focused).

The second attack generation technique we consider involves attacking the two-stage learner directly. Here, rather than using the MetaPoison technique, we consider an attack trained against a single learner which is trained from scratch at each attack epoch, giving us a more complete metagradient (computed using Higher [9]). Unlike the previous method, this one requires differentiating through the solution to an optimization problem as the two-stage learner explicitly solves that optimization problem at test time. For this component, we use Qpth [2]. Once again, after an attack is generated, we further evaluate it by testing it against the decision-focused learner.

The third and most computationally complex attack we consider is one formulated directly against the decision focused learner. On the surface, this attack is nearly identical to the one against the two-stage learner. When considered in more depth, however, it's a significantly more difficult problem, for reasons previously discussed. Thus, we are motivated to investigate the feasibility of attacking this model directly.

5.2 Experiment Domains

Synthetic Data. For our synthetic data experiments, we consider the following decision optimization problem:

$$\min f(x, \theta) = \frac{1}{2} x^T Q x - \theta^T x \;\; \text{s.t.} \;\; ||x|| \leq D, Ax \leq b \tag{14}$$

where Q is a diagonal positive-definite matrix, serving as a penalty parameter to make the problem convex, and θ is the unknown parameter that needs to be trained. $||x|| \leq D$ is simply a magnitude constraint on the decision variable, while $Ax \leq b$ represents some other constraints on the decision space. This decision optimization formulation is typically used for representing shortest path, maximum flow, bipartite matching, and a range of other domains [31].

In our experiments, in order to predict the unknown parameter θ, we consider a simple neural network and randomly (according to the normal distribution) generate synthetic data to train this predictive network. The labels are computed as a function of the features, plus a small amount of random noise. In addition, regarding the decision optimization, we randomly generate decision constraints. The amount of constraints used are varied across experiments to explore how this affects the attack generation. These constraints are added incrementally: an experiment with 9 constraints would include the same constraints as the corresponding experiment with 7 constraints, in addition to 2 new constraints.

Stock Market Portfolio Optimization. In addition to this simplified artificial problem, we demonstrate our attack in the portfolio optimization domain. This is naturally modeled via data-based decision making, where, prior to the optimization itself, future stock returns and the covariances between stocks must be predicted. This makes the domain a natural choice for our decision-focused

attack. Similar to other recent work [30], we utilize the Markowitz model [19] to maximize expected return while encouraging a diverse portfolio. Overall, the objective function of the optimization problem combines maximizing immediate return at each time step with minimizing risk, formulated as follows:

$$f(x, \theta, p, Q) = p^T x - \lambda x^T Q x$$

Where x is the investment decision made (a vector that sums to 1, representing percentage of investment in each stock), p is the expected immediate return, λ is a risk aversion parameter, and Q is a matrix capturing the covariance between the expected returns of all stocks. Intuitively, Q represents how correlated individual stocks are, and it is more risky to invest in correlated stocks. Thus, the penalty term incentivizes diverse investment.

The learning problem, then, is to utilize historical information about the stocks themselves to learn *both* the expected returns (p) as well as a 32 dimensional embedding for each stock. This embedding is then used to calculate the covariance between each pair of stocks, using cosine similarity. Specifically, we use the prices at the previous time step as well as rolling statistics as the input of the neural network to (separately) learn p and Q. As in [30] these statistics include a variety of sliding window means, as well as variances, of the historical stock prices. Loss functions for both the two-stage and the decision-focused models utilize ground-truth p and Q values directly computed from the dataset. Note that both p and Q depend on the price data from *future* timesteps: p is the next timestep's return, while Q is the cosine similarity of the returns over the next 10 timesteps.

We utilize real-world historical stock data, downloaded from the Quandl WIKI dataset, from 2004 to 2017 [25]. The stocks used belong to the SP500, giving us 505 potential stocks to work with. Attacking the features exclusively is not meaningful here, as the features are computed based on the raw price. Due to this, we target our attack on the *raw* historical stock market data, which affects the features, the labels (p), and the covariance matrix (Q). We restrict our experiments to a setting with 50 stocks and 500 timesteps.

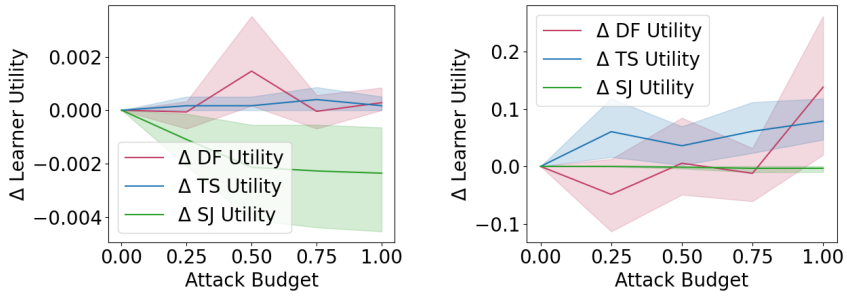
6 Results

Now we present the results of our experiments. For all the graphs, the results are averaged over 5 random seeds which determine both the initial network weights as well as the randomized attack starting points. In the synthetic data domain, this also corresponds to 5 different data sets (generated using the same normal distribution). For supplemental results, see the linked appendix[7].

6.1 Synthetic Data

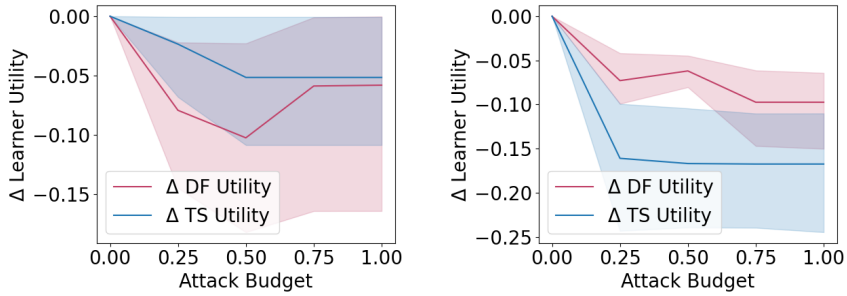
On the following graphs, a 'small dataset' refers to a setting with 250 instances in the training dataset, while a 'large dataset' refers to one with 750 elements.
Simple Joint Model. In Figure 4, we display the effectiveness of attacks gener-

[7] https://www.dropbox.com/s/6lznj4c1imk5qcm/DataBasedSupplemental.pdf



(a) Attack on Large Dataset (b) Attack on Small Dataset

Fig. 4: Attacks generated against a simple joint model.

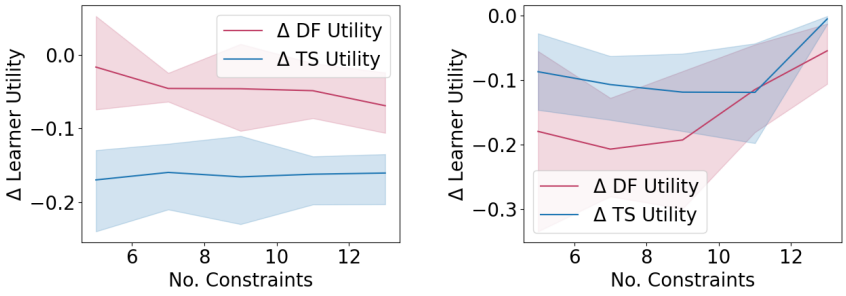


(a) Attack on Large Dataset (b) Attack on Small Dataset

Fig. 5: Attacks generated against a two-stage learner

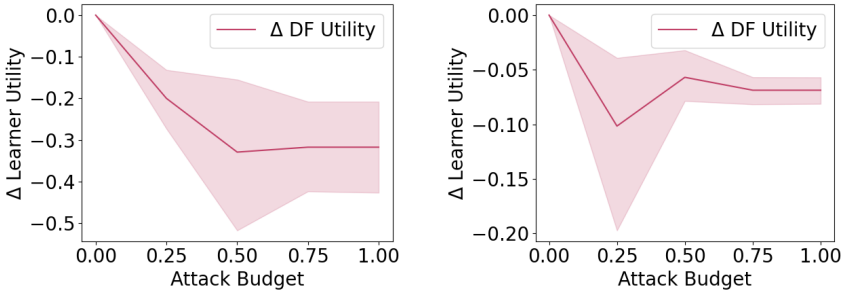
ated against a simple joint learner. Both cases here demonstrate similar trends. First, that the found attacks are only minimally effective against the simple joint learner itself. Secondly, we observe that when transferring these attacks to the decision focused and the two-stage learners the effect on their utility is inconsistent and follows no clear trends. This finding stands in contrast to the results obtained by MetaPoison in the field of computer vision [12]. While this result may be surprising, the problems being solved in data-based decision making are notably different from computer vision tasks, and the models we utilize are significantly less complex.

Two-Stage Model. In Figure 5 we show our results when generating an attack on the two-stage learner. Contrasted with the attacks in Figure 4, we observe significantly higher effectiveness, both against the two-stage learner itself and when transferring the attack to the decision-focused learner. This contrast further suggests that methods from other domains (such as computer vision) may not be directly applicable when attacking data based-decision making models.



(a) Attack on Large Dataset (b) Attack on Small Dataset

Fig. 6: Effect of adding constraints on attack results

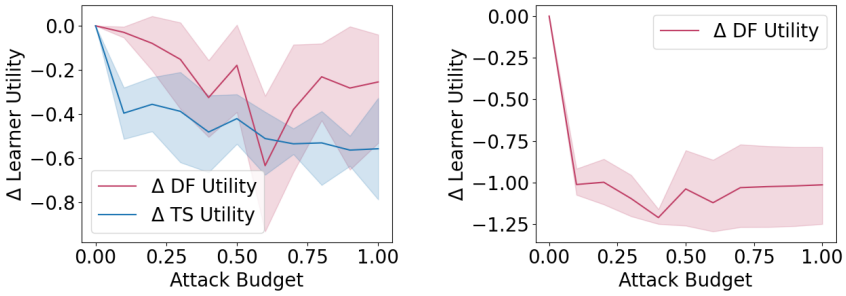


(a) Attack on Large Dataset (b) Attack on Small Dataset

Fig. 7: Attacking a decision-focused model directly

Figure 6 demonstrates the effect of introducing more constraints to the optimization problem. What we observe here is that while the effectiveness of our attacks is dependent on the constraints, there is no simple trend when varying them. This makes it hard to predict how effective a metagradient based attack will be when attacking a new problem in data-based decision making.

Decision Focused Model. In Figure 7 we examine the effectiveness of attacking a decision-focused learner directly. While our method is able to find good attacks in some scenarios, it is unreliable. Even in this simple setting, gradient descent often struggles to find a good optima, and this issue becomes even more apparent with a larger attack space. Combined with the prohibitive compute requirements of this attack, this is unlikely to be a practical approach in many data based learning settings.



(a) Attack on two-stage learner (b) Attack on decision focused learner

Fig. 8: Attacking a portfolio optimization model

6.2 Portfolio Optimization

Two-Stage Model. Figure 8a shows the results of attacking a two-stage model for portfolio optimization, as well as transferring that attack to a decision focused learner. Notably, we see that transferring the attack is often effective, though is inconsistent. This is likely due to the increased complexity of this domain compared to our synthetic setting, which includes both the transformation of raw prices into features as well as the objective of the optimization problem.
Decision Focused Model. In Figure 8b, we display the results of attacking a decision-focused model directly. In this case, this attack is on average more effective than the attacks on the two-stage model. Most of this is likely due to the decision focused learner performing better when unattacked, getting an objective value of -0.005 to -0.094, compared to the two-stage learner that obtains objective values between -0.32 and -0.65. We also observe once again that higher 'budget' attacks (meaning a larger attack space) often lead to worse attacks (higher utility for the learner), further demonstrating the complexities of solving these attacks.

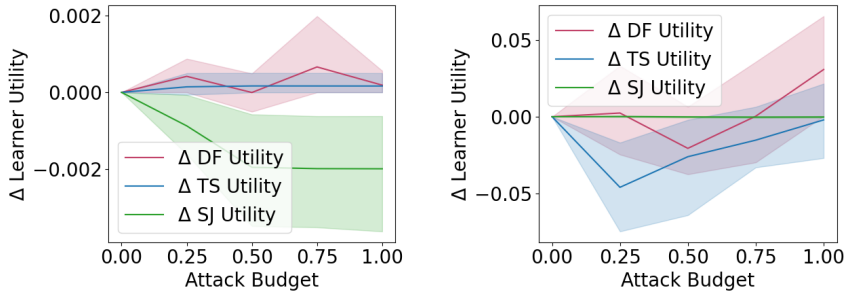
7 Conclusion

In this work, we formulated a generalized meta-gradient based poisoning attacks against two-stage models, decision focused models, and a simple joint model. We were able to provide insight into the difficulties of this attack by conducting extensive experiments in a synthetic domain as well as a real-world stock market portfolio optimization problem. These experiments show the following results. First, we observe that existing meta-gradient based techniques [12] may be ineffective here, despite being quite effective in the domain of computer vision. Next, we provide analysis showing that direct attacks on a decision-focused model are discouragingly difficult and problem dependent. Furthermore, despite the inherent training differences between two-stage and decision-focused learners, our results show that poisons crafted on a two-stage model can be effective against decision-focused models as well.

References

1. Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., Kolter, J.Z.: Differentiable convex optimization layers. Advances in neural information processing systems **32** (2019)
2. Amos, B., Kolter, J.Z.: Optnet: Differentiable optimization as a layer in neural networks. arXiv preprint arXiv:1703.00443 (2017)
3. Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., de Freitas, N.: Learning to learn by gradient descent by gradient descent (2016)
4. Bengio, Y.: Gradient-Based Optimization of Hyperparameters. Neural Computation **12**(8), 1889–1900 (08 2000). https://doi.org/10.1162/089976600300015187, https://doi.org/10.1162/089976600300015187
5. Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F.: Evasion attacks against machine learning at test time. In: Joint European conference on machine learning and knowledge discovery in databases. pp. 387–402. Springer (2013)
6. Biggio, B., Nelson, B., Laskov, P.: Poisoning attacks against support vector machines. arXiv preprint arXiv:1206.6389 (2012)
7. Donti, P., Amos, B., Kolter, J.Z.: Task-based end-to-end model learning in stochastic optimization. In: Advances in Neural Information Processing Systems. pp. 5484–5494 (2017)
8. Fang, F., Nguyen, T.H., Pickles, R., Lam, W.Y., Clements, G.R., An, B., Singh, A., Tambe, M., Lemieux, A., et al.: Deploying paws: Field optimization of the protection assistant for wildlife security. In: AAAI. vol. 16, pp. 3966–3973 (2016)
9. Grefenstette, E., Amos, B., Yarats, D., Htut, P.M., Molchanov, A., Meier, F., Kiela, D., Cho, K., Chintala, S.: Generalized inner loop meta-learning. arXiv preprint arXiv:1910.01727 (2019)
10. Huang, L., Joseph, A.D., Nelson, B., Rubinstein, B.I., Tygar, J.D.: Adversarial machine learning. In: Proceedings of the 4th ACM workshop on Security and artificial intelligence. pp. 43–58 (2011)
11. Huang, L., Joseph, A.D., Nelson, B., Rubinstein, B.I., Tygar, J.D.: Adversarial machine learning. In: AISec (2011)
12. Huang, W.R., Geiping, J., Fowl, L., Taylor, G., Goldstein, T.: Metapoison: Practical general-purpose clean-label data poisoning. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 12080–12091. Curran Associates, Inc. (2020), https://proceedings.neurips.cc/paper/2020/file/8ce6fc704072e351679ac97d4a985574-Paper.pdf
13. Jagielski, M., Oprea, A., Biggio, B., Liu, C., Nita-Rotaru, C., Li, B.: Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 19–35. IEEE (2018)
14. Karypis, G., Kumar, V.: Metis—a software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing ordering of sparse matrices (01 1997)
15. Krantz, S.G., Parks, H.R.: The implicit function theorem: history, theory, and applications. Springer Science & Business Media (2002)
16. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: Artificial intelligence safety and security, pp. 99–112. Chapman and Hall/CRC (2018)

17. Li, J., Zhang, H., Han, Z., Rong, Y., Cheng, H., Huang, J.: Adversarial attack on community detection by hiding individuals. Proceedings of The Web Conference 2020 (Apr 2020). https://doi.org/10.1145/3366423.3380171, http://dx.doi.org/10.1145/3366423.3380171
18. Lowd, D., Meek, C.: Adversarial learning. In: ACM SIGKDD (2005)
19. Markowitz, H.: Portfolio selection. The Journal of Finance **7**(1), 77–91 (1952), http://www.jstor.org/stable/2975974
20. Mukhopadhyay, A., Vorobeychik, Y.: Prioritized allocation of emergency responders based on a continuous-time incident prediction model. In: International Conference on Autonomous Agents and MultiAgent Systems (2017)
21. Muñoz-González, L., Biggio, B., Demontis, A., Paudice, A., Wongrassamee, V., Lupu, E.C., Roli, F.: Towards poisoning of deep learning algorithms with back-gradient optimization. Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (2017)
22. Papernot, N., McDaniel, P., Goodfellow, I.: Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. arXiv preprint arXiv:1605.07277 (2016)
23. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: 2016 IEEE European symposium on security and privacy (EuroS&P). pp. 372–387. IEEE (2016)
24. Perrault, A., Wilder, B., Ewing, E., Mate, A., Dilkina, B., Tambe, M.: Decision-focused learning of adversary behavior in security games (2019)
25. Quandl: WIKI various end-of-day data (2021), https://www.quandl.com/data/WIKI
26. Sen, P., Namata, G.M., Bilgic, M., Getoor, L., Gallagher, B., Eliassi-Rad, T.: Collective classification in network data. AI Magazine **29**(3), 93–106 (2008)
27. Shafahi, A., Huang, W.R., Najibi, M., Suciu, O., Studer, C., Dumitras, T., Goldstein, T.: Poison frogs! targeted clean-label poisoning attacks on neural networks. Advances in neural information processing systems **31** (2018)
28. Shah, S., Sinha, A., Varakantham, P., Perrault, A., Tambe, M.: Solving online threat screening games using constrained action space reinforcement learning. CoRR **abs/1911.08799** (2019), http://arxiv.org/abs/1911.08799
29. Wang, H., Xie, H., Qiu, L., Yang, Y.R., Zhang, Y., Greenberg, A.: Cope: Traffic engineering in dynamic networks. In: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications. pp. 99–110 (2006)
30. Wang, K., Wilder, B., Perrault, A., Tambe, M.: Automatically learning compact quality-aware surrogates for optimization problems. Advances in Neural Information Processing Systems **33**, 9586–9596 (2020)
31. Wilder, B., Dilkina, B., Tambe, M.: Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization (2018)
32. Wilder, B., Ewing, E., Dilkina, B., Tambe, M.: End to end learning and optimization on graphs (2020)
33. Xue, M., He, C., Wang, J., Liu, W.: One-to-n amp; n-to-one: Two advanced backdoor attacks against deep learning models. IEEE Transactions on Dependable and Secure Computing pp. 1–1 (2020). https://doi.org/10.1109/TDSC.2020.3028448
34. Xue, Y., Davies, I., Fink, D., Wood, C., Gomes, C.P.: Avicaching: A two stage game for bias reduction in citizen science. In: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems. pp. 776–785 (2016)
35. Zügner, D., Günnemann, S.: Adversarial attacks on graph neural networks via meta learning. In: International Conference on Learning Representations (ICLR) (2019)



(a) Portfolio Optimization (b) Bipartite Matching

Fig. 9: Attacks generated on simple joint models

8 Appendix

8.1 Experiment Domain - Bipartite Matching

Bipartite matching is a well established problem in graph learning. In form, it is essentially identical to the synthetic data setting previously discussed:

$$\min f(x, \theta) = \frac{1}{2} x^T Q x - \theta^T x \quad \text{s.t.} \quad ||x|| \leq D, Ax \leq b \tag{15}$$

In this case, however, the constraints enforced are that x must be *doubly stochastic*. Intuitively, x is a square matrix with continuous values. Each value x_{ij} represents the probability that node i on one side of the graph will be matched with node j on the other side. For the learning component of the problem, the goal is to predict the graph's edges from the nodes' features. This means that ϵ represents per-node features, while θ is the graph's adjacency matrix (relaxed to be continuous).

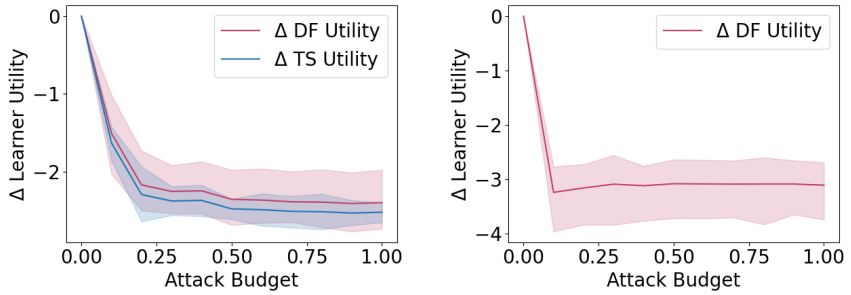
For these experiments, we utilize the Cora dataset [26] which consists of scientific papers. The features here are binary values indicating the presence of keywords in the paper, while the edges in the graph are citations. In total, there are 1433 features and 2708 nodes. Inspired by a recent paper [31], we split the dataset into 27 bipartite subgraphs, with 50 nodes on each side in each subgraph. This is accomplished using Metis [14] to partition the graph.

8.2 Supplementary Experiment Results

In Figure 9, we display the results of using Metapoison [12] to solve attacks against a simple joint learner, and transferring the found attack to the two-stage and decision focused learners. Both domains display the same trends as observed in our synthetic domain - namely, that the attack is only nominally effective against the simple joint model, and not at all effective when transferred to the other two models. Once again, this suggests that techniques from domains

such as computer vision may not be most appropriate for attacking data-based decision making models.

Figure 10 shows the results when attacking two-stage and decision focused models for bipartite matching. The trends are once again similar to the other domains: attacks trained against a two-stage learner can effectively transfer to a decision focused learner. Furthermore, as in portfolio optimization, we observe that the decision focused learner appears more susceptible to direct attack (Figure 10b) than is the two-stage learner (Figure 10a). Once again, this is likely due to the decision focused learner outperforming the two-stage counterpart in the absence of attack. Unattacked, the two-stage learner achieves utility values between 2.37 and 2.90 while the decision focused learner obtains utilities between 2.65 and 4.59. Particularly when attacking the decision focused learner (Figure 10b) we can observe the recurring trend of increased attack budgets often leading to *worse* attacks and *higher* utility for the learner, demonstrating the difficulties of finding good attack optima via (meta)gradient descent.



(a) Attack on two-stage learner (b) Attack on decision focused learner

Fig. 10: Attacking a bipartite matching model