The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)

Attacking Data Transforming Learners at Training Time

Scott Alfeld,[1] Ara Vartanian,[2] Lucas Newman-Johnson,[1] Benjamin I.P. Rubinstein[3]
[1]Department of Computer Science, Amherst College
[2]Department of Computer Sciences, University of Wisconsin – Madison
[3]School of Computing and Information Systems, University of Melbourne
[1]{salfeld, lnewmanjohnson18}@amherst.edu
[2]aravart@cs.wisc.edu
[3]brubinstein@unimelb.edu.au

Abstract

While machine learning systems are known to be vulnerable to data-manipulation attacks at both training and deployment time, little is known about how to adapt attacks when the defender transforms data prior to model estimation. We consider the setting where the defender Bob first transforms the data then learns a model from the result; Alice, the attacker, perturbs Bob's input data *prior to him transforming it*. We develop a general-purpose "plug and play" framework for gradient-based attacks based on matrix differentials, focusing on ordinary least-squares linear regression. This allows learning algorithms and data transformations to be paired and composed arbitrarily: attacks can be adapted through the use of the chain rule—analogous to backpropagation on neural network parameters—to compositional learning maps. Best-response attacks can be computed through matrix multiplications from a library of attack matrices for transformations and learners. Our treatment of linear regression extends state-of-the-art attacks at training time, by permitting the attacker to affect both features and targets optimally and simultaneously. We explore several transformations broadly used across machine learning with a driving motivation for our work being autogressive modeling. There, Bob transforms a univariate time series into a matrix of observations and vector of target values which can then be fed into standard learners. Under this learning reduction, a perturbation from Alice to a single value of the time series affects features of several data points along with target values.

Introduction

While outside the traditional focus of security, attacks on machine learning systems are inevitable: the very adaptability that learners deliver to valuable applications and industries is also an attack surface to be exploited. A key security goal in adversarial machine learning research is model integrity/availability (Barreno et al. 2006) or prediction correctness more broadly. There, a defender Bob wishes to learn and then deploy a model after the adversary Alice has perturbed train or test data. While numerous positive results demonstrate the potential power Alice might wield in lab settings (Rubinstein et al. 2009; Biggio et al. 2013; Szegedy et al. 2013), little is known about something as prevalent to practical scenarios as feature transformation and its effects on optimal attack strategies (Huang et al. 2011). This paper delivers a systematic treatment of optimal best-response attacks against a defender employing data pre-processing.

A common approach to computing data manipulation attacks is with a so-called gradient-based attack (Biggio et al. 2013), where Alice performs gradient-descent on her loss function to optimize her data perturbation. To compute the gradient we employ the theory of matrix differential calculus (MDC) (Magnus and Neudecker 1988). Manipulating differentials is more hospitable to compact calculation, though from a theoretical point of view the concepts of (matrix) differentials and (matrix) derivatives are equivalent. Through the chain rule, one can compute the gradient of Alice's loss for any transform-learner combination via matrix (or tensor) multiplication. One need only compute the (matrix) derivative of the learner and the derivative of the transform, separately. In this way, MDC yields a "plug and play" framework for generating adversarial perturbations.

A consequence is that much prior work — which focuses on settings where Alice directly affects the features or target values fed into the learner — can be leveraged to compute attacks against defenders which first transform the data. We illustrate this phenomenon by computing the derivative of ordinary least squares (OLS) regression *for arbitrary additive perturbations* and three different transforms within the MDC framework. We further demonstrate the flexibility of our framework by discussing the case when Bob performs a series of transforms prior to learning. Where prior work has often considered an attacker bound to specific types of alterations (e.g., affecting only target values or a single data point), we allow for an attacker to simultaneously alter both features and target values[1]. It is through algebraic manipulations of matrix differentials that we are able to compute the derivative of Alice's loss.

A driving motivation for our work is autoregressive (AR) modeling. Here one models a value of a time series as some

[1]Note that this does not mean our attack is unbounded in their capability. As discussed in more detail, we consider an attacker with a defined budget limiting their perturbation.

function of the prior d values of the same time series[2]. For example, suppose Alice and Bob are negotiating Bob's purchase of Alice's company. Bob might take the company's historical quarterly profits as a time series and forecast future profits. So as to get as high a price as possible, Alice wants Bob to learn a model which forecasts high profits. She might "cook the books" and manipulate her company's past profit numbers to trick Bob into learning such a model. Note that she may perform her manipulation in a legal way, by e.g., channel stuffing or earnings manipulation.

For example, Alice might reschedule sales so as to shift profits from one quarter to the following (or prior) quarter. Or she might move her costs (e.g., by changing the timing of large purchases) to quarters showing poor performance. Her hope is then that that later quarters show an improvement/recovery, which Bob may see as positive momentum. Prior work (Alfeld, Zhu, and Barford 2016) has examined deployment-time attacks against a fixed (autoregressive) model. By contrast, we compute a training time attack.

There is strong real-world motivation to study attacks against autoregressive learners due to their use in e.g., the financial sector. In addition, such models demonstrate intriguing behavior. To learn an autoregressive model, a learner can transform the time series into a collection of (feature, target) pairs which are then fed into an off-the-shelf machine learning algorithm (Bontempi, Taieb, and Le Borgne 2012). This transformation from time series to features and targets causes single values (e.g., Alice's profits for a particular quarter) to be both features and a target value. For example, suppose Bob models the profits of a quarter as some function f of the previous two quarters from a series of examples of the form $Q_i = f(Q_{i-1}, Q_{i-2})$. Altering Q_3's value will simultaneously affect three examples: $Q_3 = f(Q_2, Q_1), Q_4 = f(Q_3, Q_2), Q_5 = f(Q_4, Q_3)$. This fact that the attacker affects both features and target values distinguishes a contribution of our work from prior works in which the attacker affects only features (Jagielski et al. 2018) or only target values (Mei and Zhu 2015).

Forming a collection of (features, target) pairs from a time series is only one example of a data transformation step. Often data is e.g., centered, normalized, standardized, etc. prior to learning and if Alice performs her data manipulation attack prior to this transformation, she must account for it in selecting her attack. We argue that a realistic model of an attacker is one where she affects data prior to the transformation (which is, after all, performed by Bob).

This paper is organized as follows. First, we mathematically define the agents Alice and Bob in terms of their knowledge and goals. We then calculate the derivative of Alice's loss for OLS regression and a range of common data transformations. To empirically investigate the connection between Alice's capability and her effectiveness as an attacker, we perform experiments on synthetic data. We then discuss the most closely related prior work and conclude.

[2] d is known as the *order* or *degree* of the autoregressive model.

Problem Setup

We denote vectors by lower case bold characters (e.g., $\boldsymbol{v}, \boldsymbol{\theta}$) and matrices by upper case bold characters (e.g., $\boldsymbol{D}, \boldsymbol{H}$). We denote the i-th element of a vector as the unbolded letter with a subscript (e.g., v_i). We denote individual elements of a matrix with brackets, using colons to denote an entire row, column (e.g., $\boldsymbol{X}[:, 1]$ denotes the first column of X). We denote the $n \times n$ identity matrix as $\boldsymbol{I}_n$, and the Kronecker product by $\otimes$. Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ we denote the $mn \times 1$ vector formed by stacking $\boldsymbol{A}$'s columns as $\texttt{Vec}(\boldsymbol{A})$. We denote the $mn \times mn$ commutator matrix as $\boldsymbol{K}^{(m,n)}$ where $\boldsymbol{K}^{(m,n)}\texttt{Vec}(\boldsymbol{A}) = \texttt{Vec}(\boldsymbol{A}^\top)\ \forall \boldsymbol{A} \in \mathbb{R}^{m\times n}$.

For matrices $\boldsymbol{M} \in \mathbb{R}^{a\times b}$, and $\boldsymbol{N} \in \mathbb{R}^{c\times d}$, we define the derivative of $\boldsymbol{M}$ with respect to $\boldsymbol{N}$ using the "Good Notation" of Magnus and Neudecker (1988):

$$\nabla_{\boldsymbol{N}}\boldsymbol{M} = \frac{\mathtt{dVec}(\boldsymbol{M})}{\mathtt{dVec}(\boldsymbol{N})^\top} \in \mathbb{R}^{ab\times cd} \tag{1}$$

The i-th column of $\nabla_{\boldsymbol{N}}\boldsymbol{M}$ (which is an $ab \times 1$ vector) is $\left(\frac{\mathtt{dVec}(\boldsymbol{M})}{\mathtt{dVec}(\boldsymbol{N})_i}\right)^\top$. Note that this has the effect that when $\boldsymbol{M}$ is a column vector and $\boldsymbol{N}$ is a scalar (represented as a 1×1 matrix), then the gradient is a row vector.

Agent Definitions

We consider two agents: Alice the attacker and Bob the defender. Alice seeks to manipulate the model that Bob learns by exercising her limited capability to perturb his input data. Bob is oblivious to Alice and dutifully performs a transformation on his input data followed by a machine learning algorithm to obtain a model. We model Alice as knowing both the transformation procedure and learning algorithm Bob uses. In what follows we define the specifics of the two players.

Bob the Defender After receiving data (potentially perturbed by the attacker Alice), Bob performs a two-step process. While typically the data that Bob receives will be in the form of a matrix of features and a vector of target values, this is not always the case. To remain general and have clean notation for our later examples, we simply say there are n scalar values as input data. Bob first performs a transformation $\mathcal{T} : \mathbb{R}^n \to \mathbb{R}^{(p\times d)} \times \mathbb{R}^p$ to obtain a feature matrix $\boldsymbol{X}$ and vector[3] of target values $\boldsymbol{y}$. We denote the matrix $[\boldsymbol{y} \mid \boldsymbol{X}]$ as $\boldsymbol{D}$. As a second step, Bob feeds $\boldsymbol{X}$ and $\boldsymbol{y}$ into some fixed-length-vector learning algorithm. Specifically, Bob selects a model $\boldsymbol{\theta}^*$ from his hypothesis space Θ which minimizes his loss ℓ_B on the training data:

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} \ell_B(\boldsymbol{v}; \boldsymbol{\theta}) \tag{2}$$

For simplicity we assume a unique minimizer.

Alice the Attacker Alice can perturb the data that Bob receives (before he performs his transformation on it). Her goal is to pull Bob's learned model toward her specific target $\boldsymbol{\theta}^{\text{target}}$, captured by her loss function ℓ_A. For example, Alice

[3] We consider the case of univariate target values for notational convenience. Multidimensional targets are similar.

may have a target model which forecasts a particular pattern (e.g., a sharp increase in company profits) on some specific input (e.g., upcoming profit reports). Alice selects a vector $\boldsymbol{\delta} \in \mathbb{R}^n$ as additive noise: Bob will observe $\boldsymbol{v} + \boldsymbol{\delta}$, then learn a model on $\mathcal{T}(\boldsymbol{v} + \boldsymbol{\delta})$. We assume a powerful attacker in terms of knowledge, but bounded in ability. Namely, Alice knows $\boldsymbol{v}$ and Bob's $\mathcal{T}$ and learner mappings. However, Alice is constrained to select her attack from some set $\mathcal{C}$. This yields Alice's bi-level optimization problem:

$$\begin{aligned} \boldsymbol{\delta}^* &= \operatorname*{argmin}_{\boldsymbol{\delta}} \ell_A(\boldsymbol{\theta}^*) \qquad (3) \\ \text{s.t. } &\boldsymbol{\delta} \in \mathcal{C} \\ \boldsymbol{\theta}^* &= \operatorname*{argmin}_{\boldsymbol{\theta} \in \Theta} \ell_B(\boldsymbol{v} + \boldsymbol{\delta}; \boldsymbol{\theta}) \end{aligned}$$

Alice solves this problem using the common technique of projected gradient descent. Beginning with small random initial $\boldsymbol{\delta}^{(0)}$, she iteratively updates:

$$\boldsymbol{\delta}^{(t+1)} = \texttt{Proj}_{\boldsymbol{\delta}} \left(\boldsymbol{\delta}^{(t)} - \eta \left(\nabla_{\boldsymbol{\delta}} \ell_A \left(\boldsymbol{\theta}^{(t)} \right) \right)^\top \right) \qquad (4)$$

$$\boldsymbol{\theta}^{(t+1)} = \operatorname*{argmin}_{\boldsymbol{\theta} \in \Theta} \ell_B \left(\boldsymbol{v} + \boldsymbol{\delta}^{(t+t)}; \boldsymbol{\theta} \right) \qquad (5)$$

where η denotes her chosen step size and the function `Proj` projects its argument to the nearest[4] within $\mathcal{C}$:

$$\begin{aligned} \texttt{Proj}(\boldsymbol{\delta}') &= \operatorname*{argmin}_{\boldsymbol{\delta}} \|\boldsymbol{\delta}' - \boldsymbol{\delta}\|_2 \qquad (6) \\ \text{s.t. } &\boldsymbol{\delta} \in \mathcal{C} \end{aligned}$$

The task at hand is then to compute $\nabla_{\boldsymbol{\delta}} \ell_A(\boldsymbol{\theta})$. We use KKT conditions in a similar way to Biggio, Nelson, and Laskov (2012) and Mei and Zhu (2015). We then apply the theory of matrix differentials to compute the gradient for Alice.

Specific Instantiations Up to this point we have considered a general setting for Alice and Bob. In this paper we focus on the setting where Bob performs ordinary least-squares regression, and Alice is trying to minimize the squared deviation from Bob's resulting model to her target. That is:

$$\ell_B(\boldsymbol{v}; \boldsymbol{\theta}) = \frac{1}{2} \|\boldsymbol{X_v} \boldsymbol{\theta} - \boldsymbol{y_v}\|^2 \qquad (7)$$

$$\ell_A(\boldsymbol{\theta}) = \frac{1}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}^{\text{target}}\|^2 \qquad (8)$$

Similar methods to what we present should generalize to Bob minimizing convex regularized loss and Alice employing any differentiable loss.

There are many common examples of Bob's transformation $\mathcal{T}$, such as prepending a column of 1's to account for a bias term, lifting the original data to a higher dimension with polynomials, centering or normalizing the data and so forth. As a driving example, we consider the task of autoregression. Namely, Bob's original input is a univariate time series $\boldsymbol{v}$ of $n = |\boldsymbol{v}|$ values and he aims to learn an order-d model (using the past d values to predict the next). Bob employs the common learning reduction of forming a matrix of $p = n - d$ stacked observations (Bontempi, Taieb, and Le Borgne 2012). Given the original time series $\boldsymbol{v} = \{v_1, \ldots, v_n\}$ let

$$\boldsymbol{D_v} = \sum_{i=1} v_i \boldsymbol{H}^{(i)} \qquad (9)$$

where $\boldsymbol{H}^{(s)}$ is a $p \times (d+1)$ matrix with $\boldsymbol{H}^{(s)}_{ij} = 1$ if $s = i + j - 1$, 0 otherwise. It is convenient to define $\boldsymbol{y_v}$ as the first column of $\boldsymbol{D_v}$, and $\boldsymbol{X_v}$ as the remaining d columns of $\boldsymbol{D_v}$. This matrix and vector can then be plugged into standard (supervised) learning algorithms. We refer to this transformation as the Hankel transformation, as the resulting $\boldsymbol{D_v}$ is a Hankel matrix.

We focus on the Hankel transformation as our motivation due to three observations. First, autoregressive modeling is commonly used in the financial sector (where many actors are motivated to attack systems), while it has received relatively little attention from the adversarial learning community (notable exceptions being Alfeld, Zhu, and Barford, 2016; Alfeld, Zhu, and Barford, 2017). Second, Hankel is an example transformation in which the attacker manipulates a very different dataset from what the learner trains on. While the learner observes features and target values totaling $p \times d + p$ values, the attacker only manipulates the n values of $\boldsymbol{v}$. Third, a perturbation to even a single value of $\boldsymbol{v}$ alters both features and target values for the learner. As we discuss in more detail in Related Work, this rules out all prior methods for attacking regression as unsuitable to settings where the learner first performs a Hankel transformation. While the Hankel transformation is our driving example, other transformations fit into our framework just as easily, which we discuss below.

Illustration of the Hankel Transformation

For clarity of exposition, we illustrate the Hankel transformation with the following toy example. Consider a time series $\boldsymbol{v} = [\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}]$, and suppose Bob learns a linear (homogeneous) model of order $d = 2$. That is, the learner computes $\boldsymbol{\theta} = [\theta_1, \theta_2]^\top$ and (at deployment time) it computes $\hat{v}_i = \theta_1 v_{i-1} + \theta_2 v_{i-2}$ as its prediction. The resulting matrices are then:

$$\boldsymbol{D_v} = \begin{bmatrix} \mathtt{f} & \mathtt{e} & \mathtt{d} \\ \mathtt{e} & \mathtt{d} & \mathtt{c} \\ \mathtt{d} & \mathtt{c} & \mathtt{b} \\ \mathtt{c} & \mathtt{b} & \mathtt{a} \end{bmatrix}, \boldsymbol{X_v} = \begin{bmatrix} \mathtt{e} & \mathtt{d} \\ \mathtt{d} & \mathtt{c} \\ \mathtt{c} & \mathtt{b} \\ \mathtt{b} & \mathtt{a} \end{bmatrix}, \boldsymbol{y_v} = \begin{bmatrix} \mathtt{f} \\ \mathtt{e} \\ \mathtt{d} \\ \mathtt{c} \end{bmatrix}$$

Note that Alice perturbing value $\mathtt{c}$ will affect two rows of $\boldsymbol{X_v}$, as well as the fourth target value in $\boldsymbol{y_v}$.

Solution Methodolgy

In what follows, we compute $\nabla_{\boldsymbol{\delta}} \ell_A(\boldsymbol{\theta})$. Prior work has computed this for OLS linear regression but without a tranform step and only for special cases of the attacker, such as when Alice can only affect $\boldsymbol{X}$ (Jagielski et al. 2018) or when she can only affect $\boldsymbol{y}$ (Mei and Zhu 2015). We use the theory of matrix differentials to compute this gradient in its general

[4]While we project under the ℓ_2 norm for computational reasons, the choice is application specific.

form, allowing settings where Alice affects both features and target values simultaneously such as when the Hankel transformation is used. In addition, our solution yields a "plug and play" formulation where transformations and learning algorithms can be combined freely with one another. Through use of the chain rule, the gradient of the composition of these functions is expressed as matrix multiplication.

By the chain rule, and using the fact that $\boldsymbol{\theta}^{\text{target}}$ is constant with respect to $\boldsymbol{\delta}$:

$$\nabla_{\boldsymbol{\delta}} \ell_A(\boldsymbol{\theta}) = \left(\boldsymbol{\theta} - \boldsymbol{\theta}^{\text{target}}\right)^\top \underbrace{\nabla_{\boldsymbol{D}} \boldsymbol{\theta}}_{T_{\text{learner}}} \underbrace{\nabla_{\boldsymbol{\delta}} \boldsymbol{D}_{\boldsymbol{v}+\boldsymbol{\delta}}}_{T_{\text{transform}}} \tag{10}$$

Note that T_{learner} captures how Bob learns $\boldsymbol{\theta}$ given data, whereas $T_{\text{transform}}$ explicitly captures the fact that Alice is corrupting the original $\boldsymbol{v}$ and Bob's learner is working with the transformed version.

In what follows we compute T_{learner} for when Bob is performing ordinary least squared (OLS) regression. We then compute $T_{\text{transform}}$ for three different transformations. Note the flexibility that MDC affords; after computing T_{learner} and $T_{\text{transform}}$ for any (learner, transform) pair, they can be plugged into (10).

Matrix Differential Background

The differential of a function $F : \mathbb{R}^{n\times q} \to \mathbb{R}^{m\times p}$ is the linear term in the Taylor expansion of F. By algebraically manipulating these mathematical objects, we can compute derivatives (the derivative is the linear operator of this term) of matrix-to-matrix functions. Formally, for a point $\boldsymbol{C} \in \mathbb{R}^{n\times q}$, if there exists a real $mp \times nq$ matrix $\boldsymbol{A_C}$ such that for all $\boldsymbol{U}$:

$$\begin{aligned}&\texttt{Vec}(F(\boldsymbol{C}+\boldsymbol{U})) = \\ &\qquad \texttt{Vec}(F(\boldsymbol{C})) + \boldsymbol{A_C}\texttt{Vec}(\boldsymbol{U}) + \texttt{Vec}(R_{\boldsymbol{C}}(\boldsymbol{U}))\end{aligned} \tag{11}$$

$$\lim_{\boldsymbol{U}\to\boldsymbol{0}} \frac{R_{\boldsymbol{C}}(\boldsymbol{U})}{\|U\|} = 0 \tag{12}$$

for some function $R_{\boldsymbol{C}}$, then the (first) differential of F at $\boldsymbol{C}$ is defined by:

$$\texttt{dVec}(F) = \texttt{Vec}(\text{d}F(\boldsymbol{C};\boldsymbol{U})) = \boldsymbol{A_C}\texttt{Vec}(\boldsymbol{U}) \tag{13}$$

The above definition is adapted from (Magnus and Neudecker 1988) for notational consistency. We direct the interested reader to that book for a thorough treatment of matrix differentials.

Gradient of the Learner

To compute $\nabla_{\boldsymbol{D}}\boldsymbol{\theta}$, we make use of the KKT conditions of (2) and the implicit function theorem. $\nabla_{\boldsymbol{\theta}}\ell_B$ is a function of both $\boldsymbol{\theta}$ and $\boldsymbol{D}$ and furthermore $\nabla_{\boldsymbol{\theta}}\ell_B|_{\theta=\theta^*} = \boldsymbol{0}$. Since $\nabla\boldsymbol{\theta}\ell_B$ is continuously differentiable and $\nabla^2_{\boldsymbol{\theta}}\ell_B$ is invertible, $\nabla_{\boldsymbol{\theta}}\ell_B$ defines an implicit function $\boldsymbol{\theta}(\boldsymbol{D})$ whose Jacobian is given by:

$$T_{\text{learner}} = -\left(\nabla^2_{\boldsymbol{\theta}}\ell_B\right)^{-1}\left(\nabla_{\boldsymbol{D}}\nabla_{\boldsymbol{\theta}}\ell_B\right) \tag{14}$$

Recall that $\boldsymbol{D_v} = [\boldsymbol{y_v} \mid \boldsymbol{X_v}]$. Therefore $T_{\text{learner}} = [\nabla_{\boldsymbol{y_v}}\boldsymbol{\theta} \mid \nabla_{\boldsymbol{X_v}}\boldsymbol{\theta}]$. Differentiating each side of the matrix separately, we have

$$\nabla_{\boldsymbol{y_v}}\boldsymbol{\theta}^\top = -\left(\nabla^2_{\boldsymbol{\theta}}\ell_B\right)^{-1}\left(\nabla_{\boldsymbol{y_v}}\nabla_{\boldsymbol{\theta}}\ell_B\right) \tag{15}$$

$$\nabla_{\boldsymbol{X_v}}\boldsymbol{\theta}^\top = -\left(\nabla^2_{\boldsymbol{\theta}}\ell_B\right)^{-1}\left(\nabla_{\boldsymbol{X_v}}\nabla_{\boldsymbol{\theta}}\ell_B\right) \tag{16}$$

To compute (15), we differentiate (7) with respect to $\boldsymbol{\theta}$ and then $\boldsymbol{y_v}$ or $\boldsymbol{\theta}$ respectively, yielding:

$$\nabla_{\boldsymbol{\theta}}\ell_B = \boldsymbol{X_v}^\top\left(\boldsymbol{X_v\theta} - \boldsymbol{y_v}\right) \tag{17}$$

$$= \boldsymbol{X_v}^\top\boldsymbol{X_v\theta} - \boldsymbol{X_v}^\top\boldsymbol{y_v} \tag{18}$$

$$\Rightarrow \nabla^2_{\boldsymbol{\theta}}\ell_B = \boldsymbol{X_v}^\top\boldsymbol{X_v} \tag{19}$$

$$\Rightarrow \nabla_{\boldsymbol{y_v}}\nabla_{\boldsymbol{\theta}}\ell_B = -\boldsymbol{X_v}^\top \tag{20}$$

The first term of the product in (16) is given in (19). To compute the second term, we note that

$$\nabla_{\boldsymbol{X_v}}\nabla_{\boldsymbol{\theta}}\ell_B = \nabla_{\boldsymbol{X_v}}\left(\boldsymbol{X_v}^\top\left(\boldsymbol{X_v\theta} - \boldsymbol{y_v}\right)\right) \tag{21}$$

$$= \nabla_{\boldsymbol{X_v}}\left(\boldsymbol{X_v}^\top\boldsymbol{X_v\theta}\right) - \nabla_{\boldsymbol{X_v}}\left(\boldsymbol{X_v}^\top\boldsymbol{y_v}\right) \tag{22}$$

Up to this point we have only used standard rules of matrix calculus. To compute the two summands of (22), we utilize matrix differentials. Letting $Z = \boldsymbol{\theta}^\top \otimes \boldsymbol{I}_d$ and $Y = \boldsymbol{I}_d \otimes \boldsymbol{X_v}^\top$, we have

$$\texttt{dVec}\left(\boldsymbol{X_v}^\top\boldsymbol{X_v\theta}\right) \tag{23}$$

$$= \texttt{Vec}\left(\boldsymbol{I}_d\left(\text{d}\boldsymbol{X_v}^\top\boldsymbol{X_v}\right)\boldsymbol{\theta}\right) \tag{24}$$

$$= Z\left(\boldsymbol{K}^{(d,d)}\left(\boldsymbol{I}_d \otimes \boldsymbol{X_v}^\top\right) + Y\right)\texttt{dVec}\left(\boldsymbol{X_v}\right) \tag{25}$$

$$= Z\left(\boldsymbol{K}^{(d,d)} + \boldsymbol{I}_{d^2}\right)\left(\boldsymbol{I}_d \otimes \boldsymbol{X_v}^\top\right)\texttt{dVec}\left(\boldsymbol{X_v}\right) \tag{26}$$

$$\Rightarrow \nabla_{\boldsymbol{X_v}}\left(\boldsymbol{X_v}^\top\boldsymbol{X_v\theta}\right) = Z\left(\boldsymbol{K}^{(d,d)} + \boldsymbol{I}_{d^2}\right)Y \tag{27}$$

Note that (25) follows from the fact that $\forall \boldsymbol{A} \in \mathbb{R}^{a\times b}$, $\boldsymbol{B} \in \mathbb{R}^{c\times d}$: $\boldsymbol{K}^{(c,a)}(\boldsymbol{A}\otimes\boldsymbol{B})\boldsymbol{K}^{(b,d)} = (\boldsymbol{B}\otimes\boldsymbol{A})$ and $\forall a,b \in \mathbb{Z}_+$: $\boldsymbol{K}^{(a,b)} = (\boldsymbol{K}^{(b,a)})^{-1}$. As for the relatively simple second summand of (22), we again manipulate the matrix differential:

$$\texttt{dVec}\left(\boldsymbol{X_v}^\top\boldsymbol{y_v}\right) \tag{28}$$

$$= \left(\boldsymbol{y_v}^\top \otimes \boldsymbol{I}_d\right)\texttt{Vec}\left(\text{d}\boldsymbol{X_v}^\top\right) \tag{29}$$

$$= \left(\boldsymbol{y_v}^\top \otimes \boldsymbol{I}_d\right)\boldsymbol{K}^{(p,d)}\texttt{dVec}\left(\boldsymbol{X_v}\right) \tag{30}$$

$$= \boldsymbol{K}^{(1,d)}\left(\boldsymbol{I}_d \otimes \boldsymbol{y_v}^\top\right)\texttt{dVec}\left(\boldsymbol{X_v}\right) \tag{31}$$

$$\Rightarrow \nabla_{\boldsymbol{X_v}}\left(\boldsymbol{X_v}^\top\boldsymbol{y_v}\right) = \boldsymbol{K}^{(1,d)}\left(\boldsymbol{I}_d \otimes \boldsymbol{y_v}^\top\right) \tag{32}$$

Gradient of the Transform

In what follows, we consider three different transformations: (a) Prepending a column of 1's so as to encode for a bias term when learning, (b) lifting univariate data by appending polynomial terms, and (c) the previously discussed Hankel transformation. The first two transformations are common practice in data analysis. As we will see, the $T_{\text{transform}}$

terms for each form block-diagonal matrices. This captures the lack of interplay between perturbing features and perturbing target values, in contrast to what we observe for the Hankel transformation. We then demonstrate the power of our framework by discussing the straightforward process of composing transformations.

Prepending a Column of 1's Given p data points in $d-1$ dimensions, it's common to alter $\boldsymbol{X}$ to have an additional column of 1's on the left. To capture that the attacker can add to any element of $\boldsymbol{X}$ and/or $\boldsymbol{y}$ while keeping with our prior notation, we let $\boldsymbol{v} = \texttt{Vec}([\boldsymbol{y}^{\text{orig}} \mid \boldsymbol{X}^{\text{orig}}])$. Note that d is the new dimension (not the original) and n, the length of $\boldsymbol{\delta}$ is pd.

$$\nabla_{\boldsymbol{\delta}} \boldsymbol{D}_{\boldsymbol{v}+\boldsymbol{\delta}} = \left[\begin{array}{c|c} \boldsymbol{I}_p & \mathbf{0}_{p\times p(d-1)} \\ \hline \mathbf{0}_{p\times p} & \mathbf{0}_{p\times p(d-1)} \\ \hline \mathbf{0}_{p(d-1)\times p} & \boldsymbol{I}_{p(d-1)} \end{array}\right] \tag{33}$$

We note that this structure is intuitive. The matrix is block diagonal, and the band of 0's (with thickness p) in the lower right block captures the prepended 1's in $\boldsymbol{D}$. Namely, these bias terms are constant independent of how $\boldsymbol{\delta}$ changes.

Vandermonde Transformation To learn a univariate polynomial by solving a linear system one can lift the data, synthesizing additional features by taking powers of the original feature value. To keep with the notation of the previous sections, we let $\boldsymbol{v}$ be the original collection of (univariate) data points. Now, however, $p = n$, the number of rows in $\boldsymbol{X}_{\boldsymbol{v}}$. We denote the target values as $\boldsymbol{y}_{\boldsymbol{v}}$ as before. To learn an order $d-1$ polynomial, $\boldsymbol{D}_{\boldsymbol{v}}$ is then $[\boldsymbol{y}_{\boldsymbol{v}} \mid \boldsymbol{X}_{\boldsymbol{v}}]$ where $\boldsymbol{X}_{\boldsymbol{v}}$ is the $(p \times d)$ Vandermonde matrix:

$$\boldsymbol{X}_{\boldsymbol{v}}[i,j] = \boldsymbol{v}_i^{j-1} \tag{34}$$

We allow the attacker to affect $\boldsymbol{v}$ and/or the separate $\boldsymbol{y}_{\boldsymbol{v}}$. For ease of notation we denote these separately, and let $\boldsymbol{\delta} = [\boldsymbol{\delta}_{\boldsymbol{y}_{\boldsymbol{v}}}^\top \mid \boldsymbol{\delta}_{\boldsymbol{v}}^\top]^\top$. This yields the following block-diagonal matrix:

$$\nabla_{\boldsymbol{\delta}} \boldsymbol{D}_{\boldsymbol{v}+\boldsymbol{\delta}} = \left[\begin{array}{c|c} \boldsymbol{I}_p & \mathbf{0}_{p\times p} \\ \hline & \texttt{diag}(\boldsymbol{X}'_{\boldsymbol{v}}[:,1]) \\ \mathbf{0}_{pd\times p} & \vdots \\ & \texttt{diag}(\boldsymbol{X}'_{\boldsymbol{v}}[:,d]) \end{array}\right] \tag{35}$$

with $\boldsymbol{X}'_{\boldsymbol{v}}$ being the "component-wise derivative"of $\boldsymbol{X}_{\boldsymbol{v}}$:

$$\boldsymbol{X}'_{\boldsymbol{v}}[i,j] = \frac{\partial \boldsymbol{X}_{\boldsymbol{v}}[i,j]}{\partial v_i} \tag{36}$$

Hankel Transformation By (9) we have:

$$\boldsymbol{D}_{\boldsymbol{v}+\boldsymbol{\delta}} = \sum_{i=1} (v_i + \delta_i) \boldsymbol{H}^{(i)} \tag{37}$$

Therefore:

$$\nabla_{\boldsymbol{\delta}} \boldsymbol{D}_{\boldsymbol{v}+\boldsymbol{\delta}} = \left[\texttt{Vec}\left(\boldsymbol{H}^{(1)}\right) \;\middle|\; \ldots \;\middle|\; \texttt{Vec}\left(\boldsymbol{H}^{(n)}\right) \right] \tag{38}$$

We highlight the fact that, unlike with the prepending-1's and Vandermonde transformations, here the resulting matrix is not block diagonal. This is what captures the phenomenon where a perturbation to a single element of $\boldsymbol{v}$ affects both $\boldsymbol{X}_{\boldsymbol{v}}$ (in several rows) and $\boldsymbol{y}_{\boldsymbol{v}}$.

Composing Transformations Our framework permits arbitrary matching of transformations with learners. Moreover, the distinction between learner and transformation is purely semantic. Indeed, the learning algorithm can be thought of as a transformation from $\boldsymbol{D}$ to $\boldsymbol{\theta}$. As such, the same decomposition performed in (10) using the chain rule can be done if Bob performs a series of transformations prior to learning. For example, if Bob first performs the Hankel transformation and then the prepend-1's transformation, linear regression then learns an inhomogeneous AR model. Formally, if Bob learns a model from $\mathcal{T}_2(\mathcal{T}_1(\boldsymbol{v}))$, then the gradient T_{learner} is $\nabla_{\boldsymbol{A}} \mathcal{T}_2(\boldsymbol{A}) \nabla_{\boldsymbol{v}} \mathcal{T}_1(\boldsymbol{v})$ where $\boldsymbol{A} = \mathcal{T}_1(\boldsymbol{v})$.

Experiments

We focus our empirical experiments on the setting where Bob learns an autoregressive model. As discussed, Bob performs the Hankel transform of the time series $\boldsymbol{v}+\boldsymbol{\delta}$ after Alice has selected her attack $\boldsymbol{\delta}$. As such, prior attacks against linear regression are not applicable as they perturb either features (Jagielski et al. 2018) or target values (Mei and Zhu 2015) exclusively.

We denote Bob's learned model under $\boldsymbol{\delta} = \mathbf{0}$ (no attack by Alice) as $\hat{\boldsymbol{\theta}}$. Recall that Alice is restricted to select her attack $\boldsymbol{\delta}$ from the set $\mathcal{C}$. Note that Alice will be less (or at least no more) successful at driving Bob's learned model to her target $\boldsymbol{\theta}^{\text{target}}$ as $\mathcal{C}$ shrinks. The exact correspondence between $\mathcal{C}$ and Alice's for various targets, however, is not clear. We empirically investigate this connection through experimentation.

So as to obtain interpretable results, we let $\mathcal{C}$ be an n-ball with radius C. In the following experiments, we consider three Attackers, each bound by a different value of C. For each, we examine their effectiveness (Alice's loss) against three classes of targets. Intuitively, Alice will have more difficulty (need a larger C to obtain a small loss) when $\|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{\text{target}}\|$ is large. Thus we define the three classes of targets in terms of their distance from $\hat{\boldsymbol{\theta}}$.

In practice, when using linear autoregressive models it is common to restrict one's focus to stationary models. As such, we focus on such models here. While the exact conditions of stationarity for a model are not important here, a stationary model yields a time series which remains in statistical equilibrium. We refer to reader to work of Box et al. (2015) for a more in-depth treatment of stationarity.

To generate the synthetic series of daily values we generated 100 models of degree $d = 7$ (one week) with the following sampling procedure: We sampled model $\tilde{\boldsymbol{\theta}}$ from $\mathcal{N}(0, \boldsymbol{I}_d)$ and tested it for stationarity. Non stationary models were rejected. We then created sequences $\boldsymbol{v}^{(1)}, \ldots \boldsymbol{v}^{(100)}$, each of length $n = 100$. For each $\boldsymbol{v}^{(i)}$, we sample the first d elements iid from $\mathcal{N}(0,1)$. To construct the rest of the sequence, we iterate $\boldsymbol{v}_j^{(i)} = \tilde{\theta}_1^{(i)} \boldsymbol{v}_{j-1}^{(i)} + \ldots + \tilde{\theta}_7^{(i)} \boldsymbol{v}_{j-7}^{(i)} + \mathcal{N}(0,1)$. We then, as Bob, perform the Hankel transform on each series and train to obtain $\hat{\boldsymbol{\theta}}^{(i)}$ for $i = 1, \ldots, 100$.

To construct $\boldsymbol{\theta}^{\text{target}}$'s, we sample uniformly from d-shells of radii r_1, r_2, r_3. For each $\hat{\boldsymbol{\theta}}^{(i)}$, we construct 150 target

C	$r_1 = 1.294$	$r_2 = 1.399$	$r_3 = 1.514$
.25	0.393, 0.398	0.573, 0.578	0.722, 0.726
0.5	0.369, 0.375	0.545, 0.554	0.693, 0.702
1.0	0.333, 0.340	0.502, 0.514	0.646, 0.659
2.0	0.294, 0.299	0.446, 0.456	0.580, 0.594

Table 1: (Mean, Median) loss for attackers with constraints $\|\boldsymbol{\delta}\| \leq C_1, C_2, C_3, C_4$ and $\|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{\text{target}}\|_2 = r_1, r_2, r_3$.

models (50 in each shell). Note that it is unclear a priori what values are appropriate for radii. As such, we considered the (ℓ_2) distances between the 100 $\hat{\boldsymbol{\theta}}$'s and selected r_1 as the 5-th percentile. That is, 5% of the models are within r_1 of any particular model on average. We selected r_2 and r_3 in a similar fashion using the 25-th and 50-th percentiles respectively. This yielded $r_1 = 1.294, r_2 = 1.399, r_3 = 1.514$.

For each $\boldsymbol{v}^{(i)}$ and its associated time series and target models, we consider four attackers. Each is bound by an n-ball with radius $C_1 = 0.25, C_2, 0.5, C_3 = 1.0, C_4 = 2.0$ respectively. That is $\mathcal{C}^{(i)} = \{\boldsymbol{\delta} : \|\boldsymbol{\delta}\| \leq C_j\}$. For each of the three attackers, we computed attacks against each of the 150 targets for each of the 100 series, yielding a total of 45,000 attacks. Each attacker ran projected gradient descent with step size $\eta = .1$ and terminated when the greatest (absolute) difference between Alice's loss on the current iteration and any of the past 10 iterations was less than 1/1000. Experiments were coded using NumPy (Oliphant 2006) and run on the Google Compute Engine platform. Results are shown in Table 1.

We note several observations, the first of which is the relative scale of the r's and C's. Surprisingly, even with a relatively small change to $\|\boldsymbol{v}\|$ of up to $C_1 = .25$, Alice is able to shift Bob's learned model by $r_1 - .393 = .911$ on average when $\|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{\text{target}}\| = r_1$. Values for the other attackers are of similar scale. What is perhaps most surprising, though, is the following observation seen in each of the four attackers: As the distance from Bob's original model to Alice's target increases from r_1 to r_2 to r_3, Alice's loss increases at a much faster rate. We posit that this could be explained by either Alice finding local minima in her search process or by a more intrinsic property of her loss as a function of $\boldsymbol{\delta}$. We conjecture that if local minima were indeed the explanation then we would see a point where increasing C would stop yielding improved results. We note that C_1 is substantially smaller than r_1 while C_4 is larger than r_4, and yet we do not see such a tapering-off in Alice's loss. We therefore believe that local minima are not to blame and instead there is another, unknown, underlying phenomena at play. We intend to investigate this further in future work.

Related Work

Adversarial learning (Dalvi et al. 2004; Lowd and Meek 2005) studies the use of machine learning in the presence of intelligent adversaries. This includes data manipulation attacks at training time (as we study in this work) as well as deployment time attacks, defense strategies, and the construction of learning algorithms robust to attacks. We direct the reader to the recent surveys by Biggio and Roli (2018), and by Vorobeychik and Kantarcioglu (2018) for an overview of the field.

Most closely related to the work presented herein is the work of Mei and Zhu (2015) and Jagielski et al. (2018). Mei and Zhu also use a gradient-descent algorithm to attack linear regression, and use the same KKT trick (also used by Biggio, Nelson, and Laskov, 2012 for training time attacks against Support Vector Machines and by Cauwenberghs and Poggio, 2001 in a non-adversarial context) to convert Alice's bi-level optimization to a single-level program. In attacking linear regression at training time, however, Mei and Zhu restrict their attention to an attacker which can only affect target values, simplifying the gradient computation. In addition, their learner is not transforming the data prior to learning. Jagielski et al. attack linear regression in a setting where all target values are bounded between 0 and 1. They consider an attacker which adds arbitrary points (adhering to the bound on target values), which they capture by using the attacker adding initial points and the moving them based on the gradient of the loss.[5] Their attacker is unbounded in the sense that it may add any point (but bounded in the number of points it can add) and as such, they have no need for the projection step of our attacker's algorithm. To compute the attack, they compute the derivative of the attacker loss with respect to a single point's features (but not its target value, which their attacker leaves constant) and use a coordinate-wise gradient descent algorithm. Our work unifies and generalizes these two settings by computing the derivative of the attacker loss with respect to arbitrary additive perturbations to both features and target values. In addition, we require no bounds on the target values and can handle Bob performing a series of transformations prior to learning and after Alice's attack. This generality is required in attacking autoregressive learners, and stems from our use of MDC.

Prior work has examined attacking (and defending) autoregressive forecasters after learning has concluded at deployment time, whereas we focus on training-time attacks. Alfeld, Zhu, and Barford (2016) perform optimal deployment-time attacks against linear autoregressive models where an attacker aims to draw the defender's forecast to some target. In later work (Alfeld, Zhu, and Barford 2017), the authors defend linear models (including autoregressive forecasters) against such attackers with unknown targets by taking explicit defense actions. Separate from autoregressive models, others have investigated attacks on regression models at test time. Großhans et al. (2013) positions the interplay between attacker and defender as a Bayesian game while Tong et al. (2018) consider the setting with a single attacker and multiple learners. Our work complements prior work in that we examine attackers which perturb data during the learning phase, prior to when a model has been learned.

Related in spirit to ours is past work (Rubinstein et al. 2009; Xiao et al. 2015) which computes attacks against various forms of feature selection, and investigations of (possibly randomized) feature selection as a defense strat-

[5]We note that this same trick can be applied in our work, allowing us to model an attacker which adds points.

egy (Zhang et al. 2016; Alpcan, Rubinstein, and Leckie 2016). By contrast, ours is the first systematic approach to optimal best-response attacks to compositions of both pre-processing and learning. Pre-learning transformation has been highlighted in the literature as a potential challenge for attackers (Huang et al. 2011).

Conclusion

This paper contributes a framework based on matrix differential calculus for attacking compositions of pre-processing transformations and learning. We consider the natural threat model wherein the attacker Alice perturbs original data (prior to the defender Bob transforming it) and then Bob learns on that data.

Our use of MDC delivers two key advantages. First, the resulting plug-and-play framework permits arbitrary composition of differentiable learner and transform mappings. One can compute (matrix) derivatives of learner and transform *independently*, and then multiply the results in order to compute the derivative of the composition. As such, our framework improves the applicability of known gradient-based attacks on learners — by phrasing the known derivatives in the MDC context, attacks can be extended to affect learners which first transform the data. Second, through algebraic manipulations of a matrix differential, we computed the derivative of OLS linear regression with respect to additive changes *in both features and target values*. This extends the current state-of-the-art in training-time attacks against linear regression, which considers attackers perturbing either exclusively features or exclusively target values.

While attackers perturbing both features and target values are of independent interest of autoregressive forecasting, the Hankel transformation necessitates such an attacker model. We derive the (matrix) derivative of the Hankel transformation, as well as the transform where a learner prepends a column of 1's to account for a bias term. These, combined with the previously discussed plug-and-play nature of our framework, yield attacks against (in)homogeneous linear autoregressive learners. Similarly, we derive the (matrix) derivative of the Vandermonde transform, extending the applicability of attacks against linear regression to polynomial regression.

This work was supported by The Gregory S. Call Undergraduate Research Program at Amherst College and the Australian Research Council (DE160100584).

References

Alfeld, S.; Zhu, X.; and Barford, P. 2016. Data Poisoning Attacks against Autoregressive Models. In *Proceedings of the 30th Conference on Artificial Intelligence (AAAI 2016)*.

Alfeld, S.; Zhu, X.; and Barford, P. 2017. Explicit Defense Actions Against Test-Set Attacks. In *Proceedings of the 31th Conference on Artificial Intelligence (AAAI 2017)*, 1274–1280.

Alpcan, T.; Rubinstein, B. I. P.; and Leckie, C. 2016. Large-scale strategic games and adversarial machine learning. In *2016 IEEE 55th Conference on Decision and Control*, CDC, 4420–4426. IEEE.

Barreno, M.; Nelson, B.; Sears, R.; Joseph, A. D.; and Tygar, J. D. 2006. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, 16–25. ACM.

Biggio, B., and Roli, F. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition* 84:317–331.

Biggio, B.; Corona, I.; Maiorca, D.; Nelson, B.; Šrndić, N.; Laskov, P.; Giacinto, G.; and Roli, F. 2013. Evasion attacks against machine learning at test time. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 387–402. Springer.

Biggio, B.; Nelson, B.; and Laskov, P. 2012. Poisoning attacks against support vector machines. In Langford, J., and Pineau, J., eds., *29th International Conference on Machine Learning (ICML)*, 1807–1814. Omnipress.

Bontempi, G.; Taieb, S. B.; and Le Borgne, Y.-A. 2012. Machine learning strategies for time series forecasting. In *European Business Intelligence Summer School*, 62–77. Springer.

Box, G. E.; Jenkins, G. M.; Reinsel, G. C.; and Ljung, G. M. 2015. *Time series analysis: forecasting and control.* John Wiley & Sons.

Cauwenberghs, G., and Poggio, T. 2001. Incremental and decremental support vector machine learning. In *Advances in Neural Information Processing Systems*, 409–415.

Dalvi, N.; Domingos, P.; Sanghai, S.; Verma, D.; et al. 2004. Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 99–108. ACM.

Großhans, M.; Sawade, C.; Brückner, M.; and Scheffer, T. 2013. Bayesian Games for Adversarial Regression Problems. *Proceedings of the 30th International Conference on Machine Learning* 28:55–63.

Huang, L.; Joseph, A. D.; Nelson, B.; Rubinstein, B. I. P.; and Tygar, J. D. 2011. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, 43–58. ACM.

Jagielski, M.; Oprea, A.; Biggio, B.; Liu, C.; Nita-Rotaru, C.; and Li, B. 2018. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, 19–35.

Lowd, D., and Meek, C. 2005. Adversarial learning. In *Proceedings of the Eleventh ACM SIGKDD international Conference on Knowledge Discovery in Data Mining*, 641–647. ACM.

Magnus, J. R., and Neudecker, H. 1988. Matrix differential calculus with applications in statistics and econometrics. *Wiley Series in Probability and Mathematical Statistics*.

Mei, S., and Zhu, X. 2015. Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners. *Twenty-Ninth AAAI Conference on Artificial Intelligence* 2871–2877.

Oliphant, T. E. 2006. *A guide to NumPy*, volume 1. Trelgol Publishing USA.

Rubinstein, B. I. P.; Nelson, B.; Huang, L.; Joseph, A. D.; Lau, S.-h.; Rao, S.; Taft, N.; and Tygar, J. D. 2009. ANTIDOTE: Understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, 1–14. ACM.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Tong, L.; Yu, S.; Alfeld, S.; and Vorobeychik, Y. 2018. Adversarial regression with multiple learners. *Proceedings of the 35th International Conference on Machine Learning*.

Vorobeychik, Y., and Kantarcioglu, M. 2018. Adversarial machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* #38.

Xiao, H.; Biggio, B.; Brown, G.; Fumera, G.; Eckert, C.; and Roli, F. 2015. Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, 1689–1698.

Zhang, F.; Chan, P. P. K.; Biggio, B.; Yeung, D. S.; and Roli, F. 2016. Adversarial feature selection against evasion attacks. *IEEE Transactions on Cybernetics* 46(3):766–777.