ADVERSARIAL EXAMPLES IN CONSTRAINED DOMAINS

A Thesis in
Computer Science and Engineering
by
Ryan Sheatsley

© 2018 Ryan Sheatsley

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2018
The thesis of Ryan Sheatsley was reviewed and approved* by the following:

Patrick D. McDaniel  
William L. Weiss Professor of Information and Communications Technology  
Thesis Advisor

Trent R. Jaeger  
Professor of Electrical Engineering and Computer Science

Nicolas Papernot  
Staff Research Scientist, Google Brain  
Special Member

Chitaranjan R. Das  
Distinguished Professor of Computer Science and Engineering  
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.
Abstract

Recent advances in computer science and engineering have enabled machine learning to be at the center of many industries, including transportation, finance, healthcare, education, and even security. However, as we enter into this revolution of automation, machine learning presents its own unique list of challenges and inherent flaws. Novel research into these phenomena have unveiled adversarial examples: inputs crafted by adversaries with the intent of causing deep neural networks to misclassify. These adversarial examples present a barrier to the adoption of machine learning to the aforementioned fields, particularly security. In this paper, we present a methodology for understanding the impact of these adversarial examples in a domain, particularly network intrusion detection. Furthermore, we show that by leveraging inherent constraints enforced through these domains, the space of legitimate adversarial examples is limited, simplifying defenses against these malicious anomalies.
# Table of Contents

List of Figures .......................... v  
List of Tables .......................... vi  
Acknowledgments ........................ vii  
Dedication ............................. viii  
Thesis Statement ........................ ix  

Chapter 1  
Introduction ........................... 1

Chapter 2  
Background ............................ 3  
2.1 Methodology ........................... 5  
2.1.1 Challenges in AML with Constraints .............................. 5  
2.1.2 Crafting with Constraints .............................. 5  
2.1.3 Computing Adversarial Sketches .............................. 7  
2.2 Evaluation ............................. 10  
2.2.1 Datasets ............................. 10  
2.2.2 Experiment Overview ............................... 12  
2.2.3 Measuring Success ............................... 15  
2.3 Uncontrollable Features ............................ 19  
2.4 Discussion ............................. 22  
2.5 Conclusions ............................ 23

Appendix A ............................. 24

Bibliography ............................ 27
List of Figures

2.1 NSL-KDD model $M_A$ Perturbation Histogram produced by adversarial examples from the AJSMA in log scale - Certain features are consistently increased (a) & (b) and decreased (c), indifferent of the source class. .................................. 13
2.2 NSL-KDD class distribution for service=IRC (a) and num_access_files (b) - These two features are often perturbed due to their bias towards the “Benign” class. 14
2.3 HSG intra- and inter- transferability rates for NSL-KDD model $M_A$ for target “Benign” as a function of $L_0$ distortion $n$ - Values between 9-12 for $n$ have greater than 80% misclassification for most learning techniques. ......................... 16
2.4 AJSMA Success Rate with unperturbable features - The overall success rate starts to decrease when the adversary is limited to controlling $\sim$30% of the feature space. 20
A.1 Perturbation Histogram for MNIST (a) and Adversarial Sketches for image datasets (b). ................................................. 25
List of Tables

2.1 Model Information ................................................................. 10
2.2 NSL-KDD constraint distribution, categorized by feature type - Unlike TCP, UDP and ICMP have limited degrees of freedom. .............................. 13
2.3 Output from crafting adversarial examples with the AJSMA for NSL-KDD model, $M_A$ - Even domains with constraints are vulnerable to adversarial examples. ................................. 14
2.4 Results for AJSMA (left) and HSG (right) for all of our experiments for target class “Benign” - Values of $n$ for our sketches represent $\sim 10\%$ $L_0$ distortion. .... 17
A.1 Scikit-Learn Model Information .................................................. 24
A.2 The constraints extracted from the NSL-KDD - Unlike TCP, UDP and ICMP have limited degrees of freedom. ...................................................... 26
I would like to thank Nicolas Papernot for introducing the field of adversarial machine learning to me and the countless hours spent together on the surrounding material. You have instilled my passion for this field.

I would like to thank Raquel Alvarez for the brainstorming, and guidance through my experimentation and writing. Your feedback was immensely valuable to the formulation of this thesis.

I would like to thank Dr. Patrick McDaniel for his unwavering support, encouragement, and passion for research. You have challenged me to push myself to achieve milestones in my life that I thought were impossible. You have changed my life and without you, I would not be where I am today.

Finally, I would like to thank my mother Edda, my father Mark, and my brother Allen, for believing in me from day one. When life was difficult, you all steered me back on track and encouraged me to settle for nothing less than my dreams. I am fortunate to have a family that has loved and supported me unconditionally.
Dedication

I dedicate this thesis to my grandfather, Larry Sheatsley. You have handed down to me our keys to success: *hard work, dedication, and perseverance*. With these keys, there is nothing in this world I cannot do.
Thesis Statement

Machine learning systems deployed in constrained domains are not more robust than their unconstrained counterparts.
Chapter 1
Introduction

Machine learning algorithms are rapidly revolutionizing many industries including transportation [1], finance [2, 3], health care [4], education [5, 6], and even security [7–10]. For the past decade, we have seen a revolution in automation as research has focused on increasing the accuracy, problem size, and applicable domains for these automated learners. The results are promising: the latest learning algorithms have shown state-of-the-art accuracy for problems spanning multiple domains. However, when an adversary is introduced, the machine learning algorithms and the myriad domains they serve often become vulnerable.

The field of adversarial machine learning explores the impact of adversarial examples: inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake, e.g. misclassify [11]. Moreover, recent work has uncovered universal adversarial perturbations, where the same perturbation can be used on multiple inputs to craft adversarial examples. The existence of adversarial examples presents a compelling barrier for sensitive domains that use machine learning. Investigations have shown that no domain (thus far) is immune to this phenomenon; the scope of adversarial examples has been expansive, reaching into image processing [12–15], malware detection [16], text [17, 18] and even speech recognition [19].

A repeated criticism of adversarial machine learning research is that investigations have almost completely focused on unconstrained domains. That is, domains in which an adversary has complete control over the feature space. This stands in contrast to many practical settings (e.g., non-image-based), where adversaries are often tightly constrained by the semantics of the features and capable of only controlling a subset of features. Here, constraints are defined by the following three characteristics: values within a feature may be fixed (binary vs continuous), the values of features may be correlated (TCP flags in packets and TCP as the transport protocol), and some features may not be controllable by an adversary. Despite these observations, the algorithms that exist today are designed to attack environments that are significantly more vulnerable than practical environments. In this work, we hypothesize that systems in constrained domains are more resilient to adversarial example generation algorithms.

In this paper, we develop an augmented algorithm, the Adaptive JSMA, to construct adversarial examples that obey domain constraints. We identify and extract constraints from a dataset and integrate them when we craft adversarial examples. We design a second algorithm, the
**Histogram Sketch Generation**, the first attack to compute adversarial sketches: universal perturbations used to craft adversarial examples en masse that comply with domain constraints. With both attacks, we measure the success rate of crafting adversarial examples in both constrained and unconstrained domains (demonstrating cross-domain applicability of our approach) by attacking a model directly, mimicking existing white-box attacks. Finally, we use adversarial examples crafted from a model to attack different learning techniques trained on similar, but not identical, data, demonstrating that adversarial examples can transfer in constrained domains. In our experiments, we use two network intrusion detection datasets, the NSL-KDD and UNSW-NB15, as representatives of constrained domains and two image datasets, MNIST and the GTSRB, for unconstrained domains.

We make four contributions:

1. We introduce the **Adaptive JSMA**, which produces adversarial examples in constrained domains. After identifying constraints in a dataset, the Adaptive JSMA modifies perturbations at each iteration to comply with constraints, thus producing legal adversarial examples.

2. We introduce the **Histogram Sketch Generation**, which generates adversarial sketches: universal perturbation vectors that obey domain constraints. These sketches are constructed by applying a heuristic to perturbed features, demonstrating that constrained domains are highly vulnerable.

3. We demonstrate that if an adversary maintains attack behavior, cannot arbitrarily control certain features, and must obey domain constraints, there is still a surprising amount of exploitable attack surface to craft adversarial examples.

4. We demonstrate promising results for our two algorithms, reaching greater than 95% misclassification rates across the datasets used in our experiments. This suggests that constrained domains are as vulnerable as their unconstrained counterparts.
Chapter 2  |  Background

Adversarial machine learning research in unconstrained domains has been broad. Since the initial observations of Biggio et al. and Szegedy et al. in deep neural networks [20,21] to the robust attacks from Kurakin et al. and Sharif et al. [22,23], adversarial examples have matured from, “an intriguing property” to a tangible threat.

As the field gained traction within the security community, the first generation of attacks were formed in the context of “white-box” (or online) attacks [12–14]. Under this threat model, adversarial examples are crafted using information (e.g., model parameters) directly from the model under attack. This represents a worst-case scenario, analogous to an insider threat, since such information would not be easily accessible in most practical contexts. Naturally, this motivates a second research question: Can an adversary successfully attack a model, without having direct access to its parameters? Papernot et al. and Tramèr et al. investigated this question by leveraging transferability: an adversarial example crafted from one model will often be an adversarial example for a different model, even if they are using different training data and/or learning techniques [24,25]. Through this “black-box,” (or offline) threat model, an adversary trains a surrogate model by obtaining similar training data used in the victim model. Once complete, the surrogate model is used to craft adversarial examples which are then (with high probability) “transferred” to the victim model.

Concurrently, others have investigated what limitations (if any) exist for adversarial examples. Kurakin et al. and Brown et al. explored how adversarial examples can be applied directly to the physical domain, introducing techniques that produce adversarial examples robust to physical distortions, like rotation, scale, and other transformations [22,26]. Moreover, Goodfellow et al. and Moosavi-Dezfooli et al. analyzed universal adversarial perturbations, where the same perturbation can be used on multiple inputs to craft adversarial examples [13,27]. These universal perturbations are particularly concerning as they enable adversaries to take computation offline and amortize computational costs over many inputs. At present, we are observing an evolution in the way adversarial examples manifest, with each mutation contributing to the growing threat against deployed machine learning systems.

For this work, we modify an existing attack, the Jacobian-based Saliency Map Approach, introduced by Papernot et al. [12]. The “JSMA” produces an adversarial example by iteratively
applying perturbations to influential features in an input. The algorithm terminates when either
the input is successfully misclassified or the specified $L_0$ distance$^1$ is reached. The JSMA greedily
selects features to perturb by constructing saliency maps, which encode the influence features
have over misclassifying a particular input. This $L_0$ minimization makes the JSMA an attractive
candidate for our evaluated domain, as discussed in Section III. However, the JSMA is not a
prerequisite for crafting adversarial examples in constrained domains. Different attack algorithms
can be used, with some adjustments, as reviewed in Sections III and V. For our case study in
network intrusion detection, the JSMA was simply the most appropriate algorithm to leverage.

---

$^1$Most attack algorithms have caps on the allowable distortion they can introduce. This distortion can be
interpreted as the distance between an adversarial example and its original counterpart. There are a suite of
metrics used throughout the literature to measure this distance, which are principally $L_p$ norms [14].
2.1 Methodology

In this section, we explain the intuition behind generating adversarial examples in constrained domains and describe how to compute adversarial sketches.

2.1.1 Challenges in AML with Constraints

Crafting adversarial examples in constrained domains is a necessarily different process from crafting examples in unconstrained domains. Not all features represent the same kind of information (pixels vs packet information), nor do they describe the same kind of statistical data (discrete vs a blend of categorical, continuous, and discrete). These differences change the threat surface and the underlying assumptions surrounding the capabilities of an adversary in constrained domains. Existing algorithms are unsuitable for attacking constrained domains for these reasons:

(1) Existing algorithms are optimized for human perception. While there is an open discussion on the amount and kinds of distortion that are appropriate [14], existing algorithms have been tuned for image domains. That is, these algorithms try to minimize human perception of the perturbations inherent in adversarial examples. However, such measurements have no meaning for many constrained domains because these alterations are not perceived by humans. Therefore, using algorithms optimized for human perception offers us no utility.

(2) Existing algorithms assume adversaries have full control over the feature space. Most algorithms perturb the entire feature space to minimize an \( L_p \) norm as a surrogate for estimating a measure of human perception. This is likely an unreasonable assumption in constrained domains. For example, in network intrusion detection, features can represent broad network behaviors that exist outside the control of an adversary.

(3) Existing algorithms do not consider domain constraints. Crafting adversarial examples that obey domain constraints is necessary to mount practical attacks. As an example of a constraint, network intrusion detection datasets commonly have protocol and service (port number) as features and certain services are exclusive to certain protocols. Therefore, to produce an adversarial example that is representative of a legitimate traffic flow, algorithms need to enforce these constraints.

2.1.2 Crafting with Constraints

In this subsection, we address the constraint problem, justify our chosen distance metric, describe our intuition for the modified JSMA, and illustrate how we extract and integrate constraints.

Addressing Constraints. To address the problems identified in the prior subsection, we integrate constraint resolution into the crafting process, which guarantees that generating an adversarial example obeys not only the semantic constraints of the domain (in our context, the TCP/IP protocol), but also the probabilistic constraints of the dataset as well. As an artifact of our constraint extraction process, we may also extract constraints that are technically allowable within TCP/IP, but are simply never observed in the dataset. As a second technique, we also
simultaneously minimize the total number of features perturbed (i.e., we compute the $L_0$ norm) to facilitate the limited control an adversary may have.

**Measuring Distance.** The distance between an adversarial example and its original counterpart is a measurement of the distortion introduced by an algorithm, commonly represented by an $L_p$ norm. Most algorithms have either limits on the maximum allowable distortion or explicit termination conditions when a particular amount of distortion is realized. There is debate on the most appropriate distance measure (i.e., the choice of $L_p$ norm) to use for modeling levels of human perception [14]. Our study of constrained domains departs from this debate, as most of these domains are not inherently visual. Therefore, we argue that measuring distance under the $L_0$ norm is appropriate, particularly to address the problems listed earlier.

**Augmenting the JSMA.** By default, the JSMA parameter $\theta$ determines the magnitude and direction of a selected perturbation. A negative $\theta$ will decrease feature values and a positive $\theta$ will increase them. However, an adversary inherently has more freedom when crafting examples, and thus, we modified the JSMA to remove this limitation. This has no practical impact on the success or failure of the JSMA. Furthermore, this also gives the adversary more space to explore in a constrained domain. We refer to this modified version as the A(daptive) JSMA.

To remove this limitation, we simply evaluate both masks used in [12] that are applied to the saliency map when $\theta$ is positive or negative. Formally, for any feature $i$ to be a perturbation candidate, $i$ must satisfy:

$$
\left( \frac{\partial f_t(x)}{\partial x_i} > 0 \text{ and } \sum_{j \neq t} \frac{\partial f_j(x)}{\partial x_i} < 0 \right) \text{ or } \left( \frac{\partial f_t(x)}{\partial x_i} < 0 \text{ and } \sum_{j \neq t} \frac{\partial f_j(x)}{\partial x_i} > 0 \right)
$$

where $\frac{\partial f_t(x)}{\partial x_i}$ represents the forward derivative for a model $f$ and target class $t$ with respect to feature $i$ in an input $x$. Conceptually, the AJSMA only considers features to be perturbable if the target gradient and sum of non-target gradients are in opposing directions. Intuitively, this simply means that any perturbation reduces the distance to the target class or increases the distance to non-target classes.

This modification enables us to determine the optimal perturbation direction for increasing (the left half of the mask) or decreasing (the right half of the mask) features. Afterwards, we use the scoring metric found in [12] and return the most influential feature with the optimal perturbation direction.

**Learning Constraints.** If the constraints are not explicitly given, then they must be inferred. There are many existing algorithms that can infer constraints, i.e., hypotheses generation algorithms [19]. However, our observations of the data and domain intuition enables us to optimize the process of learning constraints through what we call primary features. These features are determinants for the legitimate values (or ranges of values) other features can have. Therefore, we structure our constraint learning algorithm around primary features.

For example, many popular network intrusion detection datasets include features that represent services (port numbers), packet flags, and other protocol-related information. Since these features share a causal relation with protocols, we designate transport layer protocols to be primary features.
features, and begin our constraint extraction there.

After primary features have been identified, we begin to extract constraints based on the following heuristic: a constraint exists between a primary feature \( k \), and any other feature \( p \), if there exists at least one input in the training set where \( k = 1 \) \& \( p \neq 0 \) (Later, we will describe the preprocessing techniques we perform on datasets to normalize data and convert categorical variables to one-hot vectors). For example, features that describe TCP packet flags would have the value 1.0 for TCP traffic flows and 0.0 for non-TCP traffic flows. Therefore, TCP packet flag features are constrained to TCP flows. Conceptually, these constraints encode the maneuvers that are possible (and probable) for an adversary.

**Integrating Constraints into the AJSMA.** Integrating constraint resolution into the AJSMA distills to removing candidate features from the search domain that violate constraints. Once a candidate feature is selected for perturbation, we check if this feature is constrained to a primary feature. This check is described by Algorithm 1.

Consider the following example using the Algorithm 1. A UDP traffic flow is given to the AJSMA. After analyzing the saliency map, the AJSMA suggests that the current service, `tftp_u`, should be switched to `ftp`, which is a service constrained to TCP. After the service switch is made, the perturbed input is presented to Algorithm 1 to check whether or not any constraint is violated. The first condition determines if the perturbed feature \( p \) is a primary feature (i.e., TCP, UDP, or ICMP). In this example, it is not, so we move to the second condition and evaluate if \( p \) is constrained to exclusively one primary feature. In this case, \( p \) is exclusively associated with TCP. Thus, the search domain is further restricted to TCP-compliant candidate features and the transport protocol of the input is switched from UDP to TCP. Since the input has switched primary features from UDP to TCP, we finally set all non-TCP features to 0. Once the AJSMA terminates, the produced adversarial example will be representative of a legal traffic flow.

### 2.1.3 Computing Adversarial Sketches

We hypothesized the existence of universal perturbations that obey domain constraints. Our experiments verified this hypothesis via adversarial sketches\(^2\): universal perturbations that obey domain constraints and also contain the following properties:

- The sketches are designed to work across domains, in that they perturb a minimal number of features (i.e., \( L_0 \)). This is unlike other image-based approaches where they perturb a continuous range of features. These approaches are not practical for non-visual domains as features do not uniformly represent the same kind of information (e.g., pixels).

- The sketches are designed for limited control over features, as they are minimized under the \( L_0 \) norm. Other approaches attempt to minimize \( L_p \), for \( p \geq 2 \), as a proxy to find

\(^2\)The concept of “sketching,” also known as *approximate query processing* [28], was first introduced by Flajolet et al. Sketching refers to a class of streaming algorithms that seek to extract information from a data stream in a single pass [29]. Commonly deployed in memory-constrained environments, these algorithms approximate or summarize the information in a given data stream. Adversarial sketches are similar to these class of algorithms as they are an approximation of a universal perturbation and computed through one pass of inputs.
Algorithm 1: Resolving Constraints

$p$ is a candidate feature, $\Gamma$ is the search domain, $S$ is the saliency map for the current input $x$, $h : K \rightarrow V$ is an associative array containing constraints.

```
Input: $p, \Gamma, S, x, h$
// $p$ is a primary feature
1 if $p \in K$ then
2 $\Gamma = \Gamma \cap h_{k\leftarrow p}$
3 $x_k \leftarrow$ switch primary feature to $k$
4 end
// $p$ is constrained to exactly one primary feature
5 else if $\exists k \in K$ where $p \in h_k$ then
6 $\Gamma = \Gamma \cap h_k$
7 $x_k \leftarrow$ switch primary feature to $k$
8 end
// $p$ is constrained to multiple primary features
9 else if $\exists k \in K$ where $p \in h_k$ then
10 $\Gamma = \Gamma - p$
// $x$ is using an illegal primary feature wrt $p$
11 if $p \notin h_k'$ where $x_{k'} = 1$ then
12 $k = \arg \max_k S_k | k \in K \land p \in h_k$
13 $\Gamma = \Gamma \cap h_k$
14 $x_k \leftarrow$ switch primary feature to $k$
15 end
16 end
// $p$ constrains all primary features
17 else if $\forall k \in K : p \in h_k$ then
18 $\Gamma = \Gamma - p$
19 end
20 if switched primary feature to $k$ then
// ensure $x$ is not using illegal features
21 $x_i \leftarrow 0 | \forall i \notin h_k$
22 end
23 return $\Gamma, x$
```

perturbations that are hard to perceive for a human. In domains where features can be semantically complex, minimizing the total number of perturbed features is an intuitive strategy.

• The sketches can illustrate influential regions of the decision surface. Other algorithms produce perturbations that visually resemble colored noise. In our image-based experiments, our sketches encode feature saliency in a manner that visually resembles the target class. We demonstrate a handful of these in the Appendix.

In our search for adversarial sketches in constrained domains, we take a principled approach using two tools: adversarial examples generated from the AJSMA and a perturbation histogram. Prior to the algorithms described earlier for computing universal adversarial perturbations [26, 27, 30], encountering these perturbations was a matter of chance: an adversary would be required to apply a perturbation generated from an attack on other unperturbed inputs and simply observe the universality of the perturbation, i.e., brute-forcing the universal perturbation [13].

Adversarial Examples from the AJSMA. Initially, we followed the naïve brute-force approach: we used adversarial examples crafted from the AJSMA to glean insights for universal perturbations. This brute-force approach is feasible for network intrusion detection datasets as their dimensionality...
and size is small. After evaluating every perturbation generated from the AJSMA on every input in the test set, we discovered a handful of universal perturbations.

**Perturbation Histograms.** The perturbation histogram encodes how perturbations are distributed en masse (an example is shown in Figure 2.1. To generate it, we simply enumerate over all of the perturbations produced by the AJSMA (which includes any additional perturbations made to resolve constraints) and record the perturbed features and directions. Our insight for the histogram is rooted in how the AJSMA saliency map scoring metric ranks influential features; we hypothesized that features commonly perturbed across inputs from different classes would be optimal candidates for building an adversarial sketch. This hypothesis was reinforced by our observation that the perturbation histogram was (relatively) static: random shuffling of partitioned training sets, unique training parameters, and random subsets of analyzed adversarial examples yielded minor changes to the perturbation histogram. These substantial adjustments to our experiment workflow demonstrated little change between perturbation histograms. These observations suggest that the perturbation histogram is a combined representation of saliency and constraints.

With the universal perturbations discovered through the AJSMA and the static nature of the perturbation histogram, we made an observation: the majority of the perturbed features (and their associated directions) in the most successful universal perturbations generated from the AJSMA mapped directly to the most perturbed features in the perturbation histogram. This key observation led us to the creation of the Histogram Sketch Generation.

**Histogram Sketch Generation.** The HSG accepts a perturbation histogram \(H\) and integer \(n\) as parameters and returns a adversarial sketch \(a\), which consists of the top \(n\) most frequently perturbed features and optimal directions\(^3\). We observed that the most successful universal perturbations generated by the AJSMA had a subset of features that directly mapped to the most frequently perturbed features in the perturbation histogram. Intuitively, if we consider these successful universal perturbations as the optimal solution, then the HSG approximates the optimal solution by returning a subset of those features (and associated directions).

It is interesting to note that the HSG is essentially the problem of variable selection in classical statistics. Variable selection involves the selection of a subset of relevant variables (or features) in the model, such that a “minimal” amount of information is lost (e.g., minimal impact on model loss). Many common procedures in classical statistics, such as LASSO or stepwise regression, aim to balance model fit with a penalty on the \(L_0\) norm (or a relaxation of this norm to \(L_1\), in the case of LASSO). Our approach of greedily selecting the top \(n\) most perturbed features is directly analogous to stepwise regression’s (greedy) selection of the \(n\) features which best explain the data. As the complexity grows combinatorially with the number of features, greedy methods are necessarily resorted to, and our approach here is no exception.

---

\(^{3}\)While the HSG does not take a target class as a parameter, it uses information directly from the perturbation histogram to create an adversarial sketch. As a consequence, the HSG will return a targeted adversarial sketch if the histogram is built from targeted adversarial examples.
2.2 Evaluation

In this section, we evaluate our approach on two network intrusion datasets, the NSL-KDD and the UNSW-NB15, as well as two image recognition datasets, the GTSRB and MNIST. Our experiments were performed on a Dell Precision T7600 with an Intel Xeon E5-2630 and a NVIDIA GeForce TITAN X. We used Cleverhans 2.0.0 [31] for creating our models and crafting adversarial examples.

In our evaluation, we answer two questions under both white- and black-box threat models.

1. Do constrained domains more robust against adversarial examples?
2. Do adversarial sketches exist in constrained domains?

Our experiments revealed that: we can craft adversarial examples with success rates greater than 95%, even in the presence of constraints; we can compute highly successful adversarial sketches, reaching greater than 80% misclassification rates for the majority of learning techniques in black-box settings.

2.2.1 Datasets

Before we describe our experiments, we provide a brief overview of the four evaluated datasets and any preprocessing that we performed. Table 2.1 describes the model architectures, hyperparameters, and model accuracies across all four datasets.

We evaluate our approach on two network intrusion detection datasets for the application of constrained domains and two image classification datasets for unconstrained domains. We use these two image classification datasets to for comparison with other works in adversarial machine learning as well as a demonstration of the cross-domain applicability of our approach. Note that these are unconstrained domains, and thus the AJSMA behaves similarly to the original JSMA (other than the fact that you can perturb in either direction).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Architecture</th>
<th>Units</th>
<th>Batch Size</th>
<th>Learning Rate</th>
<th>Epochs</th>
<th>Testing Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSL-KDD</td>
<td>MLP</td>
<td>123, 64, 32, 5</td>
<td>200</td>
<td>0.01</td>
<td>5</td>
<td>77% ± 1.0%</td>
</tr>
<tr>
<td>UNSW-NB15</td>
<td>MLP</td>
<td>196, 98, 49, 10</td>
<td>128</td>
<td>0.01</td>
<td>10</td>
<td>75% ± 1.2%</td>
</tr>
<tr>
<td>MNIST</td>
<td>CNN</td>
<td>784, 128, 128, 10</td>
<td>128</td>
<td>0.001</td>
<td>6</td>
<td>98% ± 0.1%</td>
</tr>
<tr>
<td>GTSRB</td>
<td>CNN</td>
<td>2700, 128, 128, 42</td>
<td>128</td>
<td>0.001</td>
<td>6</td>
<td>82% ± 2.0%</td>
</tr>
</tbody>
</table>

Table 2.1: Model Information

NSL-KDD. The NSL-KDD dataset is an improved variant of the KDD Cup99 dataset [32]. The KDD Cup99 (and its NSL-KDD successor) have been used widely in the network intrusion detection community. We chose to use the NSL-KDD for the novel application of adversarial examples in this field, familiarity of the dataset within the academic community, and the lack of well-formed network intrusion detection data.

The NSL-KDD contains 5 classes, with 4 attack classes and 1 benign class. It contains 125,973 samples for training and 22,543 samples for testing. It contains 41 features\(^4\), separated into four
high-level categories of features: basic features of TCP connections, content features within a connection suggested by domain knowledge, traffic features that are computed using a two-second time window, and host-based features. The NSL-KDD has been widely studied and so we defer to prior work [7] for the subtle details of the dataset.

**UNSW-NB15.** The UNSW-NB15 dataset was designed to be a modern version of the NSL-KDD, containing modern attacks that express a “low footprint” [34]. The Australian Centre for Cyber Security (ACCS) used the **IXIA PerfectStorm** tool to create a combination of normal and abnormal network traffic. The abnormal traffic generated from the IXIA PerfectStorm tool can be broken down into nine attack families, which we used as our source classes for generating our adversarial examples\(^5\). After the traffic is generated, the authors leveraged **Argus** and **Bro-IDS** tools to construct reliable features.

The UNSW-NB15 contains 10 classes, with 9 attack classes and 1 benign class. It contains 175,341 samples for training and 83,332 samples for testing. The dataset contains 48 feature, separated into four high-level categories of features: flow-based, basic connection, content, and time-based. We defer to the authors for a comprehensive description of the dataset [34] and its similarity with the NSL-KDD [35].

**MNIST.** The Modified National Institute of Standards and Technology database contains handwritten digits. We chose to use MNIST due to its simplicity, to demonstrate cross-domain applicability of our approach, and to have a direct comparison with other adversarial machine learning approaches.

The MNIST database contains 10 classes, with numerical digits from 0 through 9. It contains 60,000 samples for training and 10,000 samples for testing. Unlike the network intrusion detection datasets, no preprocessing was required to integrate this dataset into our experimental setup. We were able to use the dataset directly as-is. We defer to the authors for the intricate details surrounding the MNIST dataset [36].

**GTSRB.** The German Traffic Sign Recognition Benchmark is a dataset of common traffic signs found throughout Germany. We chose to use the GTSRB to further emphasize the cross-domain applicability of our methodology and its increased complexity over MNIST.

The GTSRB contains 42 classes. After preprocessing, our experiments contained 21,792 samples for training and 6,893 samples for testing. Throughout the dataset, there are identical images of varying sizes. We first cropped the region of interest (which contains the traffic sign) and downsampled to a final size of 30x30. For more details concerning the GTSRB, we defer to the authors [37].

\(^4\)We used the post-processing options in WEKA [33], an open-source data mining framework, to convert categorical features to one-hot vectors.

\(^5\)While the authors intended this dataset to be used for benchmarking anomaly detection algorithms, we used it as a signature-based dataset by using the last feature, “attack class,” as the label. We achieved high classification accuracies, and thus argue that our modification has no impact on the significance of our results found in the evaluation of our methodology.
2.2.2 Experiment Overview

In the following sections, we describe our experiments in detail, using the NSL-KDD as an application of our approach. In order to explore success rates of adversarial examples in constrained domains, we have four steps: data curation, constraint generation, adversarial example generation, and adversarial sketch generation.

Through our case study on network intrusion detection, we note that all of our experiments in this section emulate a realistic adversary by taking different attacks and masking them as benign traffic.

**Data Curation.** At the initial stage of our experimental pipeline, we perform two steps: shuffle and split our training sets into five parts, and perform a stratified shuffle-split of the test set into two parts. Splitting our training set into five parts lays the foundation for evaluating black-box attacks: each partition (which we refer to as A, B, C, D, E) is representative of a uniquely trained model (which we refer to as $M_A$, $M_B$, $M_C$, $M_D$, $M_E$ respectively), mirroring the setup described in [25]. This setup allows us to measure *intra-technique* transferability rates: the rate at which adversarial examples crafted from one model are also misclassified by another model with the same learning technique. Furthermore, this setup allows us to measure the converse, *inter-technique* transferability: the same misclassification rate except with a model of an entirely different learning technique.$^6$

The second step in our setup is to perform a *stratified* shuffle-split. This creates two sets of isolated inputs: we use the AJSMA on one set and the HSG on the other. This is to ensure that the sketches are only applied on inputs that had no influence in the creation of the sketch (recall that we analyze the adversarial examples created by the AJSMA to build sketches). Unlike the partitioning performed on the training set, where some models may receive few (or even zero) inputs from a particular class for training, the stratified shuffle split of the test set is to ensure that the AJSMA crafts adversarial examples from inputs spanning all classes. As all class-relevant information is distilled into the perturbation histogram, we can create effective adversarial sketches.

Using the NSL-KDD as an example for the first stage, we built a Multi-layer Perceptron with 4 layers: an input layer of 123 units, fully-connected to 64 units, fully-connected to 32 units, and finally an output layer with 5 units.$^7$ The output layer conveys our 5 classes: Normal, Probe, Denial of Service (DoS), User to Root (U2R), and Remote to Local (R2L). We used rectified linear units (ReLU) as our chosen activation function for our hidden layers and softmax at the output layer. Our models are trained via the Adam optimizer [38] with a batch size of 200 and a learning rate of 0.01 for 5 epochs. With our five splits, each model, $M_A$ through $M_E$, is trained with $\sim25,194$ inputs. With these hyperparameters, network architecture, and training set size, we were able to achieve an average $77\% \pm 1.0\%$ accuracy on the test set.

---

$^6$For our inter-technique evaluation, we consider four popular learning techniques: Logistic Regression (LR), Decision Trees (DT), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN). Each one of these learners represent different learning paradigms (and are popular in commercial and academic contexts), and are thus appropriate candidates for evaluating inter-technique transferability. Hyperparameters and other details can be found in the Appendix.

$^7$We note that the number of layers and units was influenced by research that suggests an optimal upper bound
<table>
<thead>
<tr>
<th>Feature Type</th>
<th>Protocol</th>
<th>Basic</th>
<th>Content</th>
<th>Timing-based</th>
<th>Host-based</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td></td>
<td>81</td>
<td>12</td>
<td>9</td>
<td>10</td>
<td>112</td>
</tr>
<tr>
<td>UDP</td>
<td></td>
<td>12</td>
<td>0</td>
<td>7</td>
<td>8</td>
<td>27</td>
</tr>
<tr>
<td>ICMP</td>
<td></td>
<td>14</td>
<td>0</td>
<td>7</td>
<td>8</td>
<td>29</td>
</tr>
</tbody>
</table>

Table 2.2: NSL-KDD constraint distribution, categorized by feature type - Unlike TCP, UDP and ICMP have limited degrees of freedom.

Figure 2.1: NSL-KDD model $M_A$ Perturbation Histogram produced by adversarial examples from the AJSMA in log scale - Certain features are consistently increased (a) & (b) and decreased (c), indifferent of the source class.

**Constraint Generation.** With the heuristic described in Section III, we extract constraints in the NSL-KDD with protocol as the primary feature. The intuition behind this selection is straightforward: a majority of the features in the NSL-KDD describe metadata surrounding these protocols, e.g., flag information, services, and content-related features like FTP commands. The distribution of the extracted constraints for the NSL-KDD can be found in Table 2.2. We observe that the TCP protocol offers the highest degree of maneuverability by a wide margin (and to no surprise as it constitutes the majority of traffic flows in the dataset). This is unlike UDP and ICMP, who are significantly more constrained. Table A.2 in the Appendix shows all of the constraints, sorted by type.

**Adversarial Example Generation.** Crafting adversarial examples with the AJSMA is straightforward: with the constraints integrated into the crafting process, we iterate over half of the test set for each model, $M_A$ through $M_E$, and craft adversarial examples (the other half of the test set is used to craft adversarial examples via the sketches produced by the HSG).

For the NSL-KDD, each model starts with half of the filtered test set for crafting. Table 2.3 describes the output of this stage from an example NSL-KDD run with “Benign” (0) as the target for the number hidden neurons for feed-forward networks [39]. The remainder of our hyperparameter selection follow no formal process.
AJSMA Experiment Results - Target 0 “Benign”

<table>
<thead>
<tr>
<th>Testing Inputs</th>
<th>11,272</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labeled as Target Class</td>
<td>4,855</td>
</tr>
<tr>
<td>Misclassified as Target Class</td>
<td>2,274</td>
</tr>
<tr>
<td>Number of Inputs Attacked</td>
<td>4,143</td>
</tr>
<tr>
<td>Average Distortion</td>
<td>3.14%</td>
</tr>
<tr>
<td>∼3.86 features</td>
<td></td>
</tr>
<tr>
<td>Class Success Rates</td>
<td>0:NaN, 1:97%, 2:100%, 3.99%, 4:100%</td>
</tr>
</tbody>
</table>

Table 2.3: Output from crafting adversarial examples with the AJSMA for NSL-KDD model, $M_A$ - Even domains with constraints are vulnerable to adversarial examples.

Figure 2.2: NSL-KDD class distribution for service=IRC (a) and num_access_files (b) - These two features are often perturbed due to their bias towards the “Benign” class.

We note that there are particular features that were consistently perturbed in nearly all adversarial examples, namely setting the service as IRC and increasing num_access_files. To understand this phenomenon, we used WEKA to analyze the distribution of these features, which revealed a trivial explanation: inputs that use IRC as the service and have high values for num_access_files are heavily skewed towards our target class, “Benign”. Figure 2.2 shows these distributions (Note that the “Benign” target class is bottom class (0) on the Y-axis).

**Adversarial Sketch Generation.** At the final step, we use the perturbation histogram (computed from adversarial examples crafted by the AJSMA) to create adversarial sketches. As described in Section III, the HSG creates adversarial sketches by selecting the top $n$ most perturbed features (and their directions) from the perturbation histogram. Finally, we craft
adversarial examples by applying the sketch to the second half of the test set (again, with the 
filter described in the prior stage), for each model, $M_A$ through $M_E$.

### 2.2.3 Measuring Success

With the adversarial examples crafted via the AJSMA and HSG, we measure the effectiveness 
of the two attacks through attacking models directly or through transferability. We define the 
success rate of attacking models directly (which we label as a white-box attack) as the number of 
adversarial examples misclassified as the target class over the total number of attempted inputs 
(that is, inputs in the test set that are not classified nor labeled as the target class), formally:

$$SR_{wb} = \frac{|\{x \in X : f(x) = t\}|}{|X|}$$

where $X$ represents the set of attempted inputs, $f$ is a model, and $t$ is the target class. Furthermore, 
we define the transferability success rate to be the number of adversarial examples misclassified 
as the target class by the target model over the number of successful adversarial examples crafted 
from the source model, formally:

$$SR_{transfer} = \frac{|\{x \in X : f'(x) = t\}|}{|\{x \in X : f(x) = t\}|}$$

where $X$ again represents the set of attempted inputs from the source model $f$, and $f'$ represents 
the target model.

**AJSMA-Results.** In Table 2.4 (a) on the left, we show the NSL-KDD AJSMA success rates 
for white-box attacks and transferability. The labels on the left represent the source model used 
to generate adversarial examples and the labels on top represent the target models that were 
attacked. For the intra-technique case, the white-box results can be read along the diagonal 
(as the source and target are the same model). For both intra- and inter-technique cases, the 
transferability results are represented in all other cells. Surprisingly, the AJSMA was broadly 
successful in creating targeted adversarial examples with introduced distortion comparable to 
image datasets, even in the presence of constraints.

Additionally, it is interesting that the adversarial examples produced by the AJSMA had 
notable transferability rates in intra-technique case, even with models trained on disjoint subsets 
of the training set. This would suggest that the transferability property is stronger in lower 
dimensional spaces.

**Top-n Approximation Results.** In Figure 2.3, we show NSL-KDD model $M_A$ success rates 
for intra- and inter-technique transferability for varying values of $n$ between 1 and 24. There 
are broad regions for values of $n$ which have large transferability rates, reaching near 100% 
in white-box settings and greater than 80% transferability rates for the majority of learning techniques. These results suggest the most often perturbed features (by the AJSMA) are indeed 
appropriate candidates for building an adversarial sketch.

Finally, we are surprised at the fragility of the models trained on network intrusion detection
data; sometimes only a single feature was needed to misclassify most of inputs in the test set. We believe that this fragility is partly a function of the skewed distribution certain features can have for specific classes, such as the ones shown in Figure 2.2. This insight is unlike the adversarial examples crafted in image domains, where high dimensionality and a more balanced class distribution for features appear to mitigate this fragility.

Table 2.4 show the results for both of our algorithms across all four datasets. The values of \( n \) listed equal \( \sim 10\% \) \( L_0 \) distortion for the HSG. Again, the labels on the left represent the source model used to generate adversarial examples, while the labels on top represent the target models that were attacked. For the intra-technique case, the white-box results can be read along the diagonal (as the source and target models are the same). The transferability results, for both intra- and inter-technique cases, are represented in all other cells.

We observe high success rates for both the AJSMA and the HSG in both white-box attacks and transferability for several of the datasets. While the AJSMA struggled to produce adversarial examples that transferred as the dimensionality increased, the HSG saw little loss (albeit, the HSG introduces slightly greater than double the amount of distortion of the AJSMA, which may
explain its resilience against increasing dimensionality). Furthermore, we also noticed that some models produced adversarial sketches that resulted in poor transferability. When we perform our training set split, we make no guarantees about the distribution of the classes in each split and, as a consequence, models can receive few inputs for a particular class. We hypothesize that this can produce a noisy perturbation histogram, resulting in poor transferability. We note that our MNIST experiment did not exhibit this behavior, which contains the most balanced distribution of inputs among all of the datasets. Finally, we noticed a decreasing trend in transferability of adversarial examples produced by the AJSMA (and to some extent, adversarial examples crafted the sketches) as the dimensionality of the dataset increased. This would suggest that the distance between decision boundaries appears to increase as model complexity increases, thus mitigating transferability.

<table>
<thead>
<tr>
<th>Adaptive JSMA</th>
<th>Histogram Sketch Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>n = 9</strong></td>
<td><strong>n = 24</strong></td>
</tr>
<tr>
<td>M1</td>
<td>M2</td>
</tr>
<tr>
<td>10%</td>
<td>11%</td>
</tr>
<tr>
<td>3%</td>
<td>4%</td>
</tr>
<tr>
<td>2%</td>
<td>2%</td>
</tr>
<tr>
<td>1%</td>
<td>1%</td>
</tr>
</tbody>
</table>

(a) NSL-KDD

<table>
<thead>
<tr>
<th><strong>n = 12</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
</tr>
<tr>
<td>11%</td>
</tr>
<tr>
<td>4%</td>
</tr>
<tr>
<td>2%</td>
</tr>
<tr>
<td>1%</td>
</tr>
</tbody>
</table>

(b) UNSW-NB15

<table>
<thead>
<tr>
<th><strong>n = 81</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
</tr>
<tr>
<td>11%</td>
</tr>
<tr>
<td>4%</td>
</tr>
<tr>
<td>2%</td>
</tr>
<tr>
<td>1%</td>
</tr>
</tbody>
</table>

(c) MNIST

<table>
<thead>
<tr>
<th><strong>n = 281</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
</tr>
<tr>
<td>11%</td>
</tr>
<tr>
<td>4%</td>
</tr>
<tr>
<td>2%</td>
</tr>
<tr>
<td>1%</td>
</tr>
</tbody>
</table>

(d) GTSRB

Table 2.4: Results for AJSMA (left) and HSG (right) for all of our experiments for target class “Benign” - Values of n for our sketches represent ~10% $L_0$ distortion.

From our investigation, we highlight a couple key takeaways:

1. Constraints, as they stand, are not problematic for crafting adversarial examples. The AJSMA was reached to near 100% success rates for most attempted inputs, with distortion rates comparable to image-based experiments, even in the presence of constraints.

2. Network intrusion detection data is highly fragile: small dimensionality and skewed distributions enable attack algorithms to alter very few features to successfully craft targeted adversarial examples.

3. Worst-case scenarios (i.e., white-box attacks) are highly vulnerable, not surprisingly. With
direct access to model parameters, an adversary can have a sophisticated level of control over the output of a model.

4. Transferability is a highly prevalent property in the network intrusion detection domain (more so than images). Even in the presence of disjoint training sets and different learning techniques, both attacks produced adversarial examples with surprising levels of transferability.
2.3 Uncontrollable Features

Throughout this paper, we have discussed some of the differences between adversarial machine learning in unconstrained domains and constrained domains. One of the most fundamental questions in this field is: what precisely is an “adversarial example?” There are varying definitions with different objectives within the community. In the image space, it has been generally agreed that if an attack algorithm produces perturbations that are undetectable by a human observer, then it is an adversarial example (which is reinforced by attack algorithms who have caps on the maximum allowable distortion they can introduce). However, it is not clear how this objective becomes translated in other domains.

Research outside of image space (usually) answers this question with their own definitions: perturbed malware must maintain its properties of malware [16], perturbed audio must be nearly inaudible [19], perturbed text must preserve its semantics [17,18], among other definitions. For our work in network intrusion detection, we follow an intuitive definition: perturbed network flows must maintain their attack behavior. For example, a DoS attack must still be a DoS attack post-perturbation.

However, validating attack behaviors is a nontrivial task as security is contextual: a DoS attack on a government network has different behaviors than an attack on a family business. Therefore, any sort of simulation to demonstrate attack behavior post-perturbation must be in a similar situation to the one in which the dataset was built from. This is particularly challenging for old datasets like the NSL-KDD, as even a similar network setup would have different behavior given the modern hardware and software on the systems that would comprise that network. Even if all these factors could be accounted for, there are certain classes of attacks whose “success” is challenging to measure. For example, in reconnaissance attacks it is difficult to know if the information gathered by an adversary is useful.

Regardless, all of these points of contention are driven by a single hypothesis: Perhaps adversarial examples cannot be crafted if features which represent the semantics of the attack cannot be perturbed. Instead of attempting to justify why any set of features is critical to the semantics of the attack, we take a different stance on addressing this hypothesis: even if some reasonably sized subset of features could not be perturbed (without invalidating the attack), we argue that adversarial examples can still be successfully crafted. With the lessons learned through this research, stemming from model fragility as a result of the poor underlying data, we set out to investigate this hypothesis with a simple experiment.

We train a new model on the full NSL-KDD training set and use the 100 most representative inputs of each class\(^9\) from the test set. Next, we iterate over all of the possible combinations of features an adversary wishes to leave unperturbed, i.e., \((41)^{10}\) for \(k \in \{1, 2, \ldots, 41\}\). Once

\(\text{We find the most representative inputs by maximizing the difference (via the softmax layer) between the output component that corresponds to the label and the sum of the components that represent all non-label classes.}\)

\(\text{The “R2L” class only had 17 inputs that were correctly classified. Thus, we crafted from 317 inputs as opposed to 400.}\)

\(\text{The NSL-KDD has 41 features before expanding categorical features to one-hot vectors. If a particular combination contains a categorical feature, we eliminate all possible values associated with the feature from the search domain.}\)
we have identified a set of unperturable features, we simply eliminate that set from initial search domain of the AJSMA. In this experiment, we crafted a total of 17,664,191 adversarial examples. Figure 2.4 demonstrates the success rate of the AJSMA as a function of the number of unperturable features.

The results support our argument: the success rate of AJSMA begins to decline only when extensive subsets of features are made unperturbable (around 70%). Even if an adversary has control over only 12% of the entire feature space, the success rate of crafting adversarial examples (with the most representative forms of an attack) is slightly less than 50%.

Finally, we would also like to highlight how this experiment demonstrates a type of constraint not covered in the evaluation section: features that the adversary simply does not have control over. Throughout this work, our constrains were defined via the semantics of the domain, i.e. the TCP/IP protocol. However, this experiment to preserve the semantics of the attack also serves as a demonstration of the efficacy of an adversary under this second type of constraints. These results suggest that even if an adversary maintains attack behavior and cannot arbitrarily

\[ \text{success rate} = \frac{\text{number of successful attacks}}{\text{number of possible attacks}} \]

To prevent combinatoric explosion, we randomly sampled 1,500 unique combinations if the total number of possible combinations for a particular value of $k$ exceeded 1,500. In total, we evaluated 55,723 unique combinations of unperturbable features.

Figure 2.4: AJSMA Success Rate with unperturbable features - The overall success rate starts to decrease when the adversary is limited to controlling $\sim 30\%$ of the feature space.
control certain features and must obey the TCP/IP protocol, there is still a surprising amount of exploitable attack surface to craft legal adversarial examples.
2.4 Discussion

Through our experiments in the network intrusion detection domain, we introduced a new class of algorithms designed to craft adversarial examples in the presence of constraints. Although the application of our approach was in network intrusion detection, our methodology is suitable for constrained domains in general.

While our perturbation histogram was generated by the AJSMa, our techniques are not tied to any particular attack algorithm. Our intuition is driven by examining how attack algorithms distribute perturbations, not algorithms themselves. We speculate that if we paired our approach with a more sophisticated algorithm (after some modifications to make them suitable for constrained domains, as discussed at the beginning of Section 2.1), such as the Iterative FGSM [22] or Carlini-Wagner [14], we can create even more effective adversarial sketches.

Furthermore, while we highlight the cross-domain generalization of our approach, it does not produce images that may fool a human as well as some of the other studied attacks: it is generally agreed that attack algorithms optimized under an $L_p$ norm for $p \neq 0$ produce adversarial examples that are closer visually to their benign counterparts. However, as our case study in constrained domains was inherently non-visual, we argue that the merit of our approach be based in non-visual domains, where optimizing for human perception is irrelevant.

As an aside, we also evaluated some of the most successful Adversarial Sketches produced by the HSG on empty inputs, e.g., inputs with “0” for all features. Even those inputs were successfully misclassified as the target class after applying the sketch (they were not originally classified as the target class). This would suggest that these commonly perturbed features define the intersections of the decision boundaries for most of the evaluated learning techniques.

On the topic of defense, we also hypothesize that constraints can be useful in defending against adversarial examples, even though the constraints found in the studied datasets were ineffective. Contrary to common machine learning practice, we speculate that introducing correlated (or redundant) features may offer improvements in robustness against adversarial examples at the cost of performance. The intuition is straightforward: an adversary would be required to add additional perturbations to satisfy the constraints between redundant features. Since features in domains like network intrusion detection can have significant semantics, we theorize that satisfying a multitude of constraints may degrade the effectiveness of a practical attack.

In addition, we also observe that our approach for building adversarial sketches can also be used by a defender to assess model vulnerability. In particular, a defender could design a simple mechanism (driven by the perturbation histograms) that reveals universal directions that would make the model vulnerable. The defender can then use this analysis to detect adversarial examples at deployment. However, an adversary could circumvent detection by selectively perturbing features that have less impact. Naturally, this would come at a cost of introducing additional distortion and control over more features, which may be impractical for an adversary.
2.5 Conclusions

This paper investigated the impact of adversarial examples in constrained domains. In addition to this investigation in unique domains like network intrusion detection, we introduced two new algorithms: the Adaptive JSMA, which produces adversarial examples in constrained domains, and the Top-n Approximation, which generates adversarial sketches: universal perturbations tailored to the challenges found in constrained domains. Our work demonstrates how adversaries can generate legal adversarial examples en mass that are effective in both white- and black-box threat models.

Prior to our work, the impact of adversarial learning has been largely understood in the context of unconstrained domains. We initially hypothesized that systems whose domains were constrained would be more resilient to attack algorithms. However, our investigation suggests the opposite.

Through our experiments, we observed how biased distributions coupled with low dimensionality can have a significant impact on model vulnerability, even in the presence of constraints. Furthermore, these properties appear to strengthen the effectiveness of black-box attacks.

Indeed, it remains true that no domain, however complex, is immune to adversarial machine learning. Through a simple number of transformations, an adversary can wholly control a model, thereby defeating systems deployed in sensitive domains. This reinforces the notion that this practice is a tangible threat for any system using machine learning. We emphasize the need to reevaluate the intersection between security and autonomous systems as we move towards this revolution in automation.
## Appendix

### Table A.1: Scikit-Learn Model Information

<table>
<thead>
<tr>
<th>Learning Technique</th>
<th>Parameters</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Regression</td>
<td>PENALTY C 1.0</td>
<td>74.21% 67.65% 92.02% 84.94%</td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>C 1.0, KERNEL degree 3, C 1.0, KERNEL degree 3</td>
<td>77.31% 69.02% 94.04% 84.3%</td>
</tr>
<tr>
<td>Decision Tree Classifier</td>
<td>CRITERION gini, MAX_DEPTH ∞, MIN_SAMPLES_SPLIT 2, MIN_SAMPLES_LEAF 1, MAX_FEATURES ∞</td>
<td>74.52% 73.27% 87.73% 57.20%</td>
</tr>
<tr>
<td>k-Nearest Neighbor</td>
<td>K 5, P 2</td>
<td>74.90% 72.20% 96.88% 52.83%</td>
</tr>
</tbody>
</table>
Figure A.1: Perturbation Histogram for MNIST (a) and Adversarial Sketches for image datasets (b).
Table A.2: The constraints extracted from the NSL-KDD - Unlike TCP, UDP and ICMP have limited degrees of freedom.
Bibliography


URL http://arxiv.org/abs/1712.09665


URL http://dx.doi.org/10.1016/0022-0000(85)90041-8

URL http://arxiv.org/abs/1708.05207


URL http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14