UNIVERSITY OF CALIFORNIA SAN DIEGO

Robust Machine Learning in Adversarial Setting with Provable Guarantee

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Yizhen Wang

Committee in charge:

Professor Kamalika Chaudhuri, Chair
Professor Russell Impagliazzo
Professor Tara Javidi
Professor Farinaz Koushanfar
Professor Lawrence Saul

2020

The dissertation of Yizhen Wang is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California San Diego

2020

DEDICATION

To my family.

EPIGRAPH

If you know yourself but not the enemy,
for every victory gained you will also suffer a defeat
unless you have defense with provable guarantee.
—Sun Tzu on Adversarial Machine Learning

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Kamalika Chaudhuri. During my five and a half years at UCSD, Prof. Chaudhuri has shown unreserved support and guided me to become a researcher. She teaches me how to effectively learn from literature, to clearly present my work, and most importantly, to develop ideas into research projects with perseverance. She also works with me patiently to ensure that I discover my true passion in research. I would thank Prof. Chaudhuri once again for leading me to the exciting field of adversarial machine learning.

I would also like to thank Professors Russell Impagliazzo, Tara Javidi, Farinaz Koushanfar and Lawrence Saul for serving on my thesis committee and offering many insightful comments.

I am fortunate to have collaborated with many outstanding researchers. Prof. Somesh Jha from University of Wisconsin Madison works in three of my projects and offers valuable insights from his extensive experience in system security. Mihai Christodorescu, my supervisor at Visa Research during my internship in 2018 and 2019, shares his expert knowledge in malware detection and kindly helps me engage in security seminars. Xiaozhu Meng from Rice University helps tremendously in program analysis, experiment design and model interpretability study for the robust malware detection project. I would also like to thank Shuang Song for collaborating with me in her Pufferfish privacy project, which becomes my first publication. Last, I would like to thank Yao-yuan Yang and Cyrus Rashtchian for generalizing my robust nearest neighbor analysis to a much broader setting of non-parametric learning. I am honored to have the opportunity to work with and to learn from all my collaborators.

During my time at UCSD, I have also learned a lot from interacting with fellow graduate students. I would like to thank Chicheng Zhang, Songbai Yan, Julaiti Alafate, Shuang Song, Joseph Geumlek, Shuang Liu, Zhengqin Li, Lifan Wu, all residents in Office

4142 and many others who have enlightened my research. My special gratitude goes to Chichang Zhang, who always leads by examples and sets the bar to be a researcher high. He inspires me with his rigorous attitude and immense amount of energy towards research, and will continue to be my role model in my future career.

Last, I would like to thank my family and friends for their love in my life. Graduate student life has highs and lows – your support carries me through.

Chapter 3 is based on the material in International Conference of Machine Learning 2018 (Yizhen Wang, Somesh Jha and Kamalika Chaudhuri,"Analyzing the Robustness of Nearest Neighbors to Adversarial Examples"). The dissertation author is the primary investigator and co-author of this material.

Chapter 4 is based on the manuscript submitted to Oakland Security and Privacy 2020 (Yizhen Wang, Xiaozhu Meng, Mihai Christodorescu and Somesh Jha, "Robust Machine Learning-Based Malware Detection Against Program Transformation"). The dissertation author is the co-primary investigator and co-author of these materials. This research was conducted during an internship with Visa Research from June to September in 2018 and 2019. Mahashweta Das and Mihai Christodorescu at Visa Research contributed to this research.

Chapter 5 is based on the material in Manuscript (Yizhen Wang and Kamalika Chaudhuri, "Data Poisoning Attack against Online Learning"). The dissertation author is the primary investigator and co-author of this material.

Chapter 6 is based on the material submitted to International Conference of Machine Learning 2020 (Yizhen Wang, Somesh Jha and Kamalika Chaudhuri, "An Investigation of Data Poisoning Defenses for Online Learning"). The dissertation author is the primary investigator and co-author of this material.

VITA

2014	B. S. in Computer Science, California Institute of Technology
2017	M. S. in Computer Science, University of California San Diego
2020	Ph. D. in Computer Science, University of California San Diego

PUBLICATIONS

Yizhen Wang, Xiaozhu Meng, Mihai Christodorescu and Somesh Jha, "Robust Machine Learning-Based Malware Detection Against Program Transformation", *submitted to Oakland Security and Privacy 2020*.

Yizhen Wang, Somesh Jha and Kamalika Chaudhuri, "An Investigation of Data Poisoning Defenses for Online Learning", *submitted to International Conference of Machine Learning, 2020*.

Yizhen Wang and Kamalika Chaudhuri, "Data Poisoning Attack against Online Learning", *manuscript, 2018*.

Yizhen Wang, Somesh Jha and Kamalika Chaudhuri,"Analyzing the Robustness of Nearest Neighbors to Adversarial Examples", *International Conference of Machine Learning ,2018*.

ABSTRACT OF THE DISSERTATION

Robust Machine Learning in Adversarial Setting with Provable Guarantee

by

Yizhen Wang

Doctor of Philosophy in Computer Science

University of California San Diego, 2020

Professor Kamalika Chaudhuri, Chair

Over the last decade, machine learning systems have achieved state-of-the-art performance in many fields, and are now used in increasing number of applications. However, recent research work has revealed multiple attacks to machine learning systems that significantly reduce the performance by manipulating the training or test data. As machine learning is increasingly involved in high-stake decision making processes, the robustness of machine learning systems in adversarial environment becomes a major concern.

This dissertation attempts to build machine learning systems robust to such adversarial manipulation with the emphasis on providing theoretical performance guarantees. We consider adversaries in both test and training time, and make the following contributions.

First, we study the robustness of machine learning algorithms and model to test-time adversarial examples. We analyze the distributional and finite sample robustness of nearest neighbor classification, and propose a modified 1-Nearest-Neighbor classifier that both has theoretical guarantee and empirical improvement in robustness. Second, we examine the robustness of malware detectors to program transformation. We propose novel attacks that evade existing detectors using program transformation, and then show program normalization as a provably robust defense against such transformation. Finally, we investigate data poisoning attacks and defenses for online learning, in which models update and predict over data stream in real-time. We show efficient attacks for general adversarial objectives, analyze the conditions for which filtering based defenses are effective, and provide practical guidance on choosing defense mechanisms and parameters.

Chapter 1

Introduction

The last decade has witnessed tremendous advances in machine learning as learning based methods produce new state-of-the-art performance in image recognition [KSH12, HZRS16], object detection [RDGF16, HGDG17], natural language processing [DCLT18, VSP+17] and many other fields. Machine learning products and techniques are deployed to replace human effort in security-critical domains such as automotive systems, malware detection, and finance decision-making. In these domains, failure of learning system can cause critical consequences such as traffic accidents, loss of invaluable personal data and huge monetary loss to the user. On the contrary, there are also potential adversaries – the driver's rival, a malware author and a competitor in the financial market – who benefit from the failure and thus have the incentive to actively exploit the weakness of the learning algorithm and model. It is questionable whether machine learning systems can still achieve high performance in the presence of such adversaries.

In this dissertation, we attempt to address this concern by building provably robust learning systems in adversarial setting. We focus on two security threats – the test-time attack using adversarial examples and the data poisoning attack during training time. In both threat models, the adversary wants to force the learning system to malfunction by

tampering the data used in a machine learning pipeline. In the former setting, the adversary perturbs the test input to fool the model, while in the latter, it poisons the training data so that the learning algorithm obtains a sub-optimal model. Given the adversary's capability to manipulate these data, we analyze the adversarial impact on undefended models, and find defense mechanisms that bounds the adversarial impact with guarantees.

The dissertation is organized as follows. Chapter 2 presents main related work in adversarial machine learning and also learning scenarios involved in the dissertation. Chapter 3 contains our work analyzing the robustness of nearest neighbor classification against test-time adversaries. Chapter 4 presents our work on robust malware detection against program transformation. Chapter 5 presents data poisoning attack against online learners. Chapter 6 investigates data poisoning defenses for online learning.

Adversarial Examples and Test-time Adversarial Attack

In test-time attack, an adversary has the ability to provide modified test inputs to an already-trained classifier, but cannot modify the training process or the model in any way. A test-time adversary profits from the mistakes of the victim machine learning model made over modified test inputs. Test-time attack has received a lot of recent attention because of the discovery of *adversarial examples*. [GSS14, PMJ+16, SZS+13, PMG+17] finds that the adversary only needs to perturb legitimate test inputs by a "small amount" in order to force the classifier to report an incorrect label, and the resulted perturbed test inputs are called adversarial examples. An example is an adversary that replaces a stop sign by a slightly defaced version in order to force an autonomous vehicle to recognize it as an yield sign. The adversarial examples are similar to the original legitimate inputs by human standard and thus have the same ground truth label, yet will be classified differently.

The existence of adversarial examples causes serious concern over the reliablity and safety of machine learning systems because most machine learning models, ranging from deep neural nets to nonparametric models such as decison trees and nearest neighbors, are all susceptible to adversarial examples. Meanwhile, the adversarial perturbation can often be imperceptible to human, which makes detection hard. While the initial attack is for image classification tasks in which the attacker can perturb the pixel values and has full knowledge of the victim model, subsequent work on adversarial attack has extended the attack paradigm to more domains such as voice recognition, text sentiment analysis and malware detection, to physical object instead of digital input, and to grey-box and the black-box settings where the attacker not necessarily have full knowledge of the victim model. With commercial products reportedly being compromised by adversarial attacks, building models robust to adversarial examples becomes one of the top challenges to machine learning community.

Early defense mechanisms simulate potential attackers in training and optimize models on a mixture of clean and adversarially created training instances. Such approach yields increased robustness against existing attacks, but often fails against new attacks. In order to end the arms-race between attack and defense, recent work on adversarial examples starts to investigate the causes of adversarial examples and to develops defenses with provable guarantees. Our work in Chapter 3 is one of first paper that analyzes model robustness to L_p-norm bounded perturbation in large sample limit and in finite sample case. Inspired by the analysis, we propose novel nearest neighbor algorithms with provable guarantee.

While a large body of attack and defense are on L_p-norm bounded adversarial perturbation on input vectors, recent work also starts to consider more types of adversarial perturbation. Defenses on L_p-norm bounded perturbation lead to models robust to adversarial examples with similar *syntactic* representation to the original legitimate test

inputs. However, in many applications, a good model should also be robust to adversarial examples with similar *semantics* as the original input. For example, a face authentication tool ideally should be able to recognize the user's identity even if the camera angle is tilted or the ambient lighting has slightly changed. In Chapter 4, we investigate machine learning models in malware detection, a scenario in which the attacker – the malware author – can tranform the malware freely at the syntax level as long as the malicious functionality is maintained. We examine the threats to current models and propose new defenses with provable guarantee.

Data Poisoning Attacks

A major factor behind recent advances of machine learning models is the availabilty of big data. On the other hand, checking the integrity of such high volume data obtained in the wild using human experts becomes infeasible. It is then possible that the attacker can inject poisoning examples to the training set to reduce the performance of the model.

In data poisoning attacks, an adversary is aware of the learner's training data and algorithm, and has the power to alter a small fraction of the training data in order to make the trained classifier satisfy certain objectives. For example, a sabotage adversary of an industrial pipeline may hijack sensors and send false observation to degrade the performance of the control policy learned from the observation, or an online store may hire malicious reviewers to report false preferences so that the resulting recommendation model favors its product over its competitor's.

There has been a body of prior work on data poisoning with increasingly sophisticated attacks and defenses [BNL12, MZ15, MGBD$^+$17, SHN$^+$18, KSL18, SKL17, LWSV16, PMGGL18]. However, most work is on the offline batch setting – all data is provided in advance and the adversary assumes that the learner's goal is to produce an empirical mini-

mizer of a loss. This assumption excludes many modern learning settings that are online in nature, i.e. data arrives sequentially, the environment may change and timely model updates are required. For example, in malware detection, new malware appears everyday so the model needs to constantly adapt to new patterns and be updated incrementally. In financial industry, trading and transaction data keeps arriving and the volume is so large that storing and training on all history data is prohibitive for real-time responses. In privacy-aware setting such as federated learning, data is stored on distributed devices and the learner can only obtain sequential model updates from individual users. In contrast to extensive studies for the offline setting, little is understood in the context of online learning.

In this dissertation, we focus on data poisoning in the online setting to bridge the gap. We find the threat of data poisoning for online learning challenging as there exist highly effective attacks that can achieve its objective with a small number of poisoning examples. In order to address the problem, we formally show the conditions for which popular filtering based defense mechanisms are effective, and also give guidance to choosing the best defense mechanism in practice.

Our Contribution

Robust Nearest Neighbor Classification with Provable Guarantees

In Chapter 3, we introduce a theoretical framework analogous to bias-variance theory for understanding the sources of model robustness. Following the framework, we analyze the robustness of k nearest neighbor classification to L_p-norm constrained adversarial examples in large sample limit and with finite training data. Our analysis shows that its robustness properties depend critically on the value of k – the classifier may be inherently non-robust for small k, but its robustness approaches that of the Bayes Optimal classifier for fast-growing k. We then propose a modified 1-nearest-neighbor algorithm with provable

robustness guarantee in large sample limit. The proposed 1-NN classifier also achieves significantly higher robustness than vanilla nearest neighbors in experiments on various data sets against both white-box and black-box attacks.

Robust Malware Detection under Adversarial Program Transformation

In Chapter 4, we attempt to address the challenge of malware detection against program transformation – a process in which the malware author changes the program syntax while still maintains the original semantics or key malicious funciton. We formalize potential transformation to program syntax as logic relations, and propose a provably robust approach of learning on normalized inputs. In theory, we compare two approaches – program normalization and robust optimization – in terms of their objectives, robustness guarantees, model capacity requirement and hardness in training. In practice, we launch two novel API-substitution attacks against detectors using neural net models, and validate the gain in robustness on real world data sets using program normalization.

Effectiveness Analysis of Data Poisoning Defenses for Online Learning

In Chapter 5 and 6, we investigate data poisoning attacks and defenses for online learning, in which data arrives in stream and the model makes real-time prediction. In Chapter 5, we introduce two general attack objectives for online learning – semi-online and fully-online – and design computationally efficient attack algorithms for both objectives. These highly effective attacks against learners using the popular online gradient descent algorithm shows the security threats and calls for effective defenses. In Chapter 6, we analyze the conditions for which defense mechanisms are effective or fragile for general filtering based defense mechanisms, and instantiate the analysis for L_2-norm based defenses and two popular data-dependent defenses – L_2-distance-to-centroid and Slab. The results are validated on various real-world data sets, and can serve as guidelines for practitioners

in choosing defense mechanism and parameters.

Chapter 2

Related Work

2.1 Adversarial Examples, Test-time Adversarial Attack and Defenses

2.1.1 Under L_p-norm Constraints

[SZS+13] first finds adversarial examples to neural net models with small L_p-norm of perturbation by solving a contrained optimization problem. Since then, there has a plethora of work on finding adversarial examples. [GSS14] proposes the first gradient based method, the fast gradient sign method (FGSM), which is able to generate adversarial examples using a single round of gradient computation. [MDFF15] and [MMS+17] further improve the gradient-based methods to generate adversarial examples of smaller L_p-norm of perturbation than using [GSS14]'s method within comparable time. Meanwhile, adversarial attacks are also extended to indifferentiable, non-parametric models, as [ABE+16, PMG16] successfully find adversarial examples for nearest neighbors and decision tree.

In addition to improving white-box attacks where the attackers have full knowledge of the model parameters and outputs, another important line of work focuses on more

realistic threat models, in which the attackers only have limited knowledge of the models and limited access to model outputs. [PMG16] finds adversarial examples "transferable" – the adversarial examples to one model can also be used to attack a similar model. Such transferability enables black-box attacks against unknown models by crafting adversarial examples for the attacker's own substitute models on the same task. [CZS+17] proposes attacks using only the zeroth order output of the model as the attacker often do not have access to gradient information in real-world applications. [PYZ18] further lowers the cost of black-box attack by reducing the number of queries to the victim models. Work on these threat models also implies that security by obscurity is not an effective approach against adversarial attacks.

The security threats have motivated a large volume of work on the defense end. [MMS+18] formulates the defense into a min-max optimization problem and proposes a principled approach – adversarial training – to increase the model's robustness to adversarial examples. [MC17] focus on detecting adversarial examples via learning the manifold of clean inputs using labeled and unlabeled data. Work including [HA17, ZYJ+19] adds regularization terms to the empirical risk minimization learning process, where the regularizers are associated with model robustness. Lastly, [LAG+19] uses differential privacy techniques to ensure the model's outputs are insensitive to changes in inputs. These approaches do increase the model robustness to various extent. However, they either are broken by new attacks or give up too much accuracy on clean inputs, and hence the arms race between attacker and defender continues.

Instead of developing increasingly sophisticated empirical attacks and defenses, our work in Chapter 3 is one of the first to provide theoretical insights on the cause of adversarial examples. We introduce a theoretical framework analogous to bias-variance theory for understanding the impact of the underlying data distribution and sample size to model robustness. We use our framework to analyze the robustness of a canonical

non-parametric classifier – the k-nearest neighbors, and proposes a novel 1-nearest neighbor classifier with robustness guarantees in large sample limit. The robustness of the modified 1-NN algorithm is also validated against both white-box and black-box attacks.

2.1.2 Beyond L_p-norm Constraints

For image recognition, [ETT+17] raises the question that L_p-norm constraints on perturbation is not sufficient to capture all adversarial images that are visually similar to the original legitimate input. [HP18] has found successful attacks against image classifiers by rotation and change of backgrounds, which still keeps the semantic meaning of the image by human perception. The study of semantic-preserving attacks quickly extends to natural language processing, as [ERLD17, YCH+18, LWC+19, ZSAL19, HSW+19] find that some state-of-the-art models are not robust to word substitution using synonym.

One domain that has been severely threatened by semantic-preserving adversarial examples is system security, as [GPM+17, ADHHO18, RSRE18, HT18, RSER19] find that machine learning based malware detectors fails when the attacker adds dummy calls in program and running traces. Such vulnerability also leads to the discussion of whether machine learning should be used in the security domain [SMK+18].

While current defenses has lead to empirically more robust defense to specific attacks, there is a general lacking of provably robust defense mechanism. In Chapter 4, we look closely into semantic-preserving examples for malware detection over binary encoded input feature vectors. We formalize the adversarial action as logical transformation, and try two approaches – program normalization and robust optimization – to enhance model's robustness. We show that the normalization approach can provably enhance model robustness against adversarial attack using program transformation, and find that both approaches have good empirical performances.

2.2 Data Poisoning Attack and Defenses

2.2.1 Attacks

Prior work on data poisoning has mostly looked at offline learning algorithms. This includes attacks on spam filters [NBC+08] and malware detection [NKS06] to more recent work on sentiment analysis [NPXNR14] and collaborative filtering [LWSV16]. [BNL12] developed the first gradient-ascent-based poisoning attack on offline SVMs; this technique was later extended to neural networks [MGBD+17]. Mei and Zhu [MZ15] propose a general optimization framework for poisoning attacks in the offline setting; they show how to use KKT conditions to efficiently calculate the gradients for gradient ascent.

In contrast, little is known about poisoning attacks in more online and adaptive settings prior to our work in Chapter 5. [BL17] invesigates the effect of poisoning over data stream on the final classifier of online learning algorithms; however, the poisoning goal is limited to misclassify a pre-specified input. [AZB16] attacks an autoregressive model to yield desired outputs by altering the initial input; however their attack does not alter the model parameters, nor does it have access to the training process. Steinhardt *et al.* [SKL17] uses an online learning framework to generate a poisoning data set, but their attack and certified defense both apply to offline learning algorithms. Lastly, [HPG+17] and [LHL+17] consider attacks against an reinforcement learning agent. However, the attacks only provide adversarial examples at test time using gradient methods; the policies are already learned and the attacker has no access to the training process.

In Chapter 5, we initiate a systematic investigation of data poisoning attack for online learning. We formalize two general settings – semi-online and fully-online – which covers a wide range of online learning scenarios including [BL17]'s. We propose gradient-based computationally efficient algorithms to find the poisoning examples in both settings. These optimization problems are considerably more complex than than in [BNL12] and

[MZ15] because the KKT conditions in [MZ15] no longer apply. We address the challenges with novel optimization processes.

2.2.2 Defenses

Previous work on data poisoning defenses has attempted to provide theoretical bound of the effectiveness of poisoning attack as well as provide practical learning algorithms that deter the attacks. [DKK+18b] provides a convex optimization algorithm that is resistant to data poisoning attacks and has provable guarantees. [SKL17] bounds the effectiveness of offline data poisoning attacks using online learning losses. [JOB+18] proposes a robust regression learning algorithm that removes potential poisoning examples based on their losses over the fitted model. These results, however, only applies to the offline learning scenario, as the algorithms requires knowledge of the entire data sets, which is unavailable to the learner in the online setting. Meanwhile, there has also been prior work on outlier detection by [DKS16, DKK+17, DKK+18a]; however, these are largely in the unsupervised learning setting.

To the best of our knowledge, our work in Chapter 6 is the first to provide theoretical performance guarantees of attacks and defenses for online data poisoning. We analyze the conditions under which online poisoning attack is rapid or impossible for various defense mechanisms, and validate the effectiveness of popular data-dependent defense mechanisms on a number real-world data sets.

2.3 Security and Program Analysis

2.3.1 Machine-Learning Based Malware Detection

Machine-learning techniques have recently been widely used for malware detection and reportedly achieved high performance over inputs of both static program analysis and

dynamic traces [ASH+14, HS16]. Unfortunately such state-of-the-art malware detectors are still vulnerable to attacks that adaptively transform the malware or its dynamic behavior while keeping the malicious functionality [GPM+17, ADHHO18, RSRE18, HT18, RSER19]. [GPM+17, ADHHO18] attack binary API/library usage based neural net detectors by adding dummy API calls and library import on the feature level, and then enhance the model's robustness using adversarial training. Similarly, [RSRE18, HT18] attacks dynamic traces based recurrent neural net detectors via inserting API calls into the program traces. On the defense side, [ITAW18] enforces robustness against API additions by using monotonic classifiers.

Compared to [GPM+17, ADHHO18, RSRE18, HT18, ITAW18], our work in Chapter 4 consider model robustness against more general set of adversarial actions beyond simple addition. We formally define possible adversarial actions as transformations that respect sets of logical relations, which can readily incorporate other program obfuscation actions such as API substitution and re-implementation. The work that also considers subsitution is [RSER19], which groups API calls with similar embedding and squeezes similar calls in dynamic traces. While [RSER19] takes a statistical approach of learning the grouping from data, our work instead focuses on enforcing known logical relations in the model, and the relation can be more general than similarity between single API calls.

2.3.2 Program Normalization

Normalization was introduced to network security in the early 2000s in the context of network intrusion detection systems [HPK01] and later applied to malware detection [CJK+07]. In both settings, normalization is a technique to reduce the number of syntactically distinct instances that need to be considered by the detection system. This is particularly important in security systems that are based on rules, as operating on normalized instances immediately translates to fewer classification rules and presumedly

better accuracy as the class of malicious items can be covered by rules. A particular subset of normalization, known as *deobfuscation*, targets the evasion techniques commonly used by malware writers: virtualization [CLD11], layout obfuscation [BRTV16], renaming obfuscation [BPM17], and metadata hiding [SB16].

In Chapter 4, we address the open problem whether normalization is useful in the context of machine learning-based malware detection. We show the security guarantee and performance trade-off using feature normalization and validate the results on real-world data set.

2.4 Learning Related

2.4.1 Nearest Neighbor Classification

There has been a body of work on the convergence and consistency of nearest-neighbor classifiers and their many variants [CH67, Sto77, KP95, DW77, CD14, KW15]; all these works however consider accuracy and not robustness.

In the asymptotic regime, [CH67] shows that the accuracy of 1-nearest neighbors converges in the large sample limit to $1-2R^*(1-R^*)$ where R^* is the error rate of the Bayes Optimal classifier. This implies that even 1-nearest neighbor may achieve relatively high accuracy even when $p(y=1|x)$ is not 0 or 1. In contrast, we show that 1-nearest neighbor is *inherently non-robust* when $p(y=1|x) \in (0,1)$ under some continuity conditions.

For larger k, the accuracy of k-nearest neighbors is known to converge to that of the Bayes Optimal classifier if $k_n \to \infty$ and $k_n/n \to 0$ as the sample size $n \to \infty$. We show that the robustness also converges to that of the Bayes Optimal classifier when k_n grows at a much higher rate – fast enough to ensure uniform convergence. Whether this high rate is necessary remains an intriguing open question.

Finite sample rates on the accuracy of nearest neighbors are known to depend

heavily on properties of the data distribution, and there is no distribution free rate as in parametric methods [DW77]. [CD14] provides a clean characterization of the finite sample rates of nearest neighbors as a function of natural *interiors* of the classes. Here we build on their results by defining a stricter, more robust version of interiors and providing bounds as functions of these new robust quantities.

In Chapter 3, we propose a novel 1-nearest-neighbor algorithm to increase the robustness in the presence of test-time adversaries. We also analyze the large sample limit performance of the proposed algorithm.

2.4.2 Model Interpretability

Another related field in machine learning is model interpretability, which attempts to explain machine learning model' decision and extract knowledge from model [RSG16, STY17, GMR+19]. In Chapter 4, we use the method in [STY17] to interpret our learned model as a sanity check to see if our model's basis of prediction matches human expert's expectation.

Chapter 3

Analyzing the Robustness of Nearest Neighbor to Adversarial Examples

3.1 Introduction

Motivated by safety-critical applications, test-time attacks on classifiers via adversarial examples has recently received a great deal of attention. However, there is a general lack of understanding on why adversarial examples arise; whether they originate due to inherent properties of data or due to lack of training samples remains ill-understood.

In this chapter, we develop a theoretical framework for robust learning in order to understand the effects of distributional properties and finite samples on robustness. Building on traditional bias-variance theory [FHT00], we posit that a classification algorithm may be robust to adversarial examples due to three reasons. First, it may be *distributionally robust*, in the sense that the output classifier is robust as the number of training samples grow to infinity. Second, even the output of a distributionally robust classification algorithm may be vulnerable due to too few training samples – this is characterized by finite sample robustness. Finally, different training algorithms might result in classifiers with different

degrees of robustness, which we call *algorithmic robustness*. These quantities are analogous to bias, variance and algorithmic effects respectively.

Next, we analyze a simple non-parametric classification algorithm: *k-nearest neighbors* in our framework. Our analysis demonstrates that large sample robustness properties of this algorithm depend very much on k.

Specifically, we identify two distinct regimes for k with vastly different robustness properties. When k is constant, we show that k-nearest neighbors has zero robustness in the large sample limit in regions where $p(y = 1|x)$ lies in $(0, 1)$. This is in contrast with accuracy, which may be quite high in these regions. For $k = \Omega(\sqrt{dn \log n})$, where d is the data dimension and n is the sample size, we show that the robustness region of k-nearest neighbors approaches that of the Bayes Optimal classifier in the large sample limit. This is again in contrast with accuracy, where convergence to the Bayes Optimal accuracy is known for a much slower growing k [DGKL94, CD14].

Since $k = \Omega(\sqrt{dn \log n})$ is too high to use in practice with nearest neighbors, we next propose a novel robust version of the 1-nearest neighbor classifier that operates on a modified training set. We provably show that in the large sample limit, this algorithm has superior robustness to standard 1-nearest neighbors for data distributions with certain properties.

Finally, we validate our theoretical results by empirically evaluating our algorithm on three datasets against several popular attacks. Our experiments demonstrate that our algorithm performs better than or about as well as both standard 1-nearest neighbors and nearest neighbors with adversarial training – a popular and effective defense mechanism. This suggests that although our performance guarantees hold in the large sample limit, our algorithm may have good robustness properties even for realistic training data sizes.

3.2 The Setting

3.2.1 The Basic Setup

We consider test-time attacks in a white box setting, where the adversary has full knowledge of the training process – namely, the type of classifier used, the training data and any parameters – but cannot modify training in any way.

Given an input x, the adversary's goal is to perturb it so as to force the trained classifier f to report a different label than $f(x)$. The amount of perturbation is measured by an application-specific metric d, and is constrained to be within a radius r. Our analysis can be extended to any metric, but for this chapter we assume that d is the Euclidean distance for mathematical simplicity; we also focus on binary classification, and leave extensions to multiclass for future work.

Finally, we assume that unlabeled instances are drawn from an instance space $\mathcal{X}$, and their labels are drawn from the label space $\{0, 1\}$. There is an underlying data distribution $\mathcal{D}$ that generates labeled examples; the marginal over $\mathcal{X}$ of $\mathcal{D}$ is μ and the conditional distribution of labels given x is denoted by η.

3.2.2 Robustness and Astuteness

We begin by defining robustness, which for a classifier f at input x is measured by the robustness radius.

Definition 3.2.1 (Robustness Radius)**.** *The robustness radius of a classifier f at an instance $x \in \mathcal{X}$, denoted by $\rho(f, x)$, is the shortest distance between x and an input x' to which f assigns a label different from $f(x)$:*

$$\rho(f, x) = \inf_r \{\exists x' \in \mathcal{X} \cap B(x, r) \ s.t \ f(x) \neq f(x')\}$$

Observe that the robustness radius measures a classifier's local robustness. A classifier f with robustness radius r at x guarantees that no adversarial example of x with norm of perturbation less than r can be created using any attack method. A plausible way to extend this into a global notion is to require a lower bound on the robustness radius everywhere; however, only the constant classifier will satisfy this condition. Instead, we consider robustness around *meaningful instances*, that we model as examples drawn from the underlying data distribution.

Definition 3.2.2 (Robustness with respect to a Distribution)**.** *The robustness of a classifier f at radius r with respect to a distribution μ over the instance space $\mathcal{X}$, denoted by $R(f, r, \mu)$, is the fraction of instances drawn from μ for which the robustness radius is greater than or equal to r.*

$$R(f, r, \mu) = \Pr_{x \sim \mu}(\rho(f, x) \geq r)$$

Finally, observe that we are interested in classifiers that are *both* robust and accurate. This leads to the notion of astuteness, which measures the fraction of instances on which a classifier is both accurate and robust.

Definition 3.2.3 (Astuteness)**.** *The astuteness of a classifier f with respect to a data distribution $\mathcal{D}$ and a radius r is the fraction of examples on which it is accurate and has robustness radius at least r; formally,*

$$Ast_{\mathcal{D}}(f, r) = \Pr_{(x,y) \sim D}(\rho(f, x) \geq r, f(x) = y),$$

Observe that astuteness is analogous to classification accuracy, and we argue that it is a more appropriate metric if we are concerned with both robustness and accuracy. Unlike accuracy, astuteness cannot be directly empirically measured unless we have a way to certify a lower bound on the robustness radius. In this work, we will prove bounds on

the astuteness of classifiers, and in our experiments, we will approximate it by measuring resistance to standard attacks.

3.2.3 Sources of Robustness

There are three plausible reasons why classifiers lack robustness – *distributional, finite sample* and *algorithmic*. These sources are analogous to bias, variance, and algorithmic effects respectively in standard bias-variance theory.

Distributional robustness measures the effect of the data distribution on robustness when an infinitely large number of samples are used to train the classifier. Formally, if S_n is a training sample of size n drawn from $\mathcal{D}$ and $A(S_n, \cdot)$ is a classifier obtained by applying the training procedure A on S_n, then the distributional robustness at radius r is $\lim_{n\to\infty} \mathbb{E}_{S_n\sim\mathcal{D}}[R(A(S_n, \cdot), r, \mu)]$.

In contrast, for finite sample robustness, we characterize the behaviour of $R(A(S_n, \cdot), r, \mu)$ for finite n – usually by putting high probability bounds over the training set. Thus, finite sample robustness depends on the training set size n, and quantifies how it changes with sample size. Finally, robustness also depends on the training algorithm itself; for example, some variants of nearest neighbors may have higher robustness than nearest neighbors itself.

3.2.4 Nearest Neighbor and Bayes Optimal Classifiers

Given a training set $S_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ and a test example x, we use the notation $X^{(i)}(x)$ to denote the i-th nearest neighbor of x in S_n, and $Y^{(i)}(x)$ to denote the label of $X^{(i)}(x)$.

Given a test example x, the k-nearest neighbor classifier $A_k(S_n, x)$ outputs:

$$= 1, \quad \text{if } Y^{(1)}(x) + \ldots + Y^{(k)}(x) \geq k/2$$
$$= 0, \quad \text{otherwise.}$$

The Bayes optimal classifier g over a data distribution D has the following classification rule:

$$g(x) = \begin{cases} 1 & \text{if } \eta(x) = \Pr(y = 1|x) \geq 1/2; \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

3.3 Robustness of Nearest Neighbors

How robust is the k-nearest neighbor classifier? We show that it depends on the value of k. Specifically, we identify two distinct regimes – constant k and $k = \Omega(\sqrt{dn \log n})$ where d is the data dimension – and show that nearest neighbors has different robustness properties in the two.

3.3.1 Low k Regime

In this region, k is a constant that does not depend on the training set size n. Provided certain regularity conditions hold, we show that k-nearest neighbors is inherently non-robust in this regime unless $\eta(x) \in \{0, 1\}$ – in the sense that the distributional robustness becomes 0 in the large sample limit.

Theorem 3.3.1. *Let $x \in \mathcal{X} \cap supp(\mu)$ such that (a) μ is absolutely continuous with respect to the Lebesgue measure (b) $\eta(x) \in (0, 1)$ (c) η is continuous with respect to the Euclidean metric in a neighborhood of x. Then, for fixed k, $\rho(A_k(S_n, \cdot), x)$ converges in probability to 0.*

Remarks. Observe that Theorem 3.3.1 implies that the distributional robustness (and hence astuteness) in a region where $\eta(x) \in (0,1)$ is 0. This is in contrast with accuracy; for 1-NN, the accuracy converges to $1 - 2R^*(1 - R^*)$ as $n \to \infty$, where R^* is the error rate of the Bayes Optimal classifier, and thus may be quite high.

The proof of Theorem 3.3.1 in the Appendix shows that the absolute continuity of μ with respect to the Lebesgue measure is not strictly necessary; absolute continuity with respect to an embedded manifold will give the same result, but will result in a more complex proof.

In the Appendix A (Theorem A.2), we show that k-nearest neighbor is astute in the interior of the region where $\eta \in \{0,1\}$, and provide finite sample rates for this case.

3.3.2 High k Regime

Prior work has shown that in the large sample limit, the accuracy of the nearest neighbor classifiers converge to the Bayes Optimal, provided k is set properly. We next show that if k is $\Omega(\sqrt{dn \log n})$, the regions of robustness and the astuteness of the k nearest neighbor classifiers also approach the corresponding quantities for the Bayes Optimal classifier as $n \to \infty$. Thus, if the Bayes Optimal classifier is robust, then so is k-nearest neighbors in the large sample limit.

The main intuition is that $k = \Omega(\sqrt{dn \log n})$ is large enough for *uniform convergence* – where, with high probability, *all* Euclidean balls with k examples have the property that the empirical averages of their labels are close to their expectations. This guarantees that for any x, the k-nearest neighbor reports the same label as the Bayes Optimal classifier for *all* x' close to x. Thus, if the Bayes Optimal classifier is robust, so is nearest neighbors.

Definitions

We begin with some definitions that we can use to characterize the robustness of the Bayes Optimal classifier. Following [CD14], we use the notation $B^o(x, r)$ to denote an open ball and $B(x, r)$ to denote a closed ball of radius r around x. We define the probability radius of a ball around x as:

$$r_p(x) = \inf\{r \mid \mu(B(x, r)) \geq p\}$$

We next define the r-robust (p, Δ)-strict interiors as follows:

$$\begin{aligned}
\mathcal{X}^+_{r,\Delta,p} &= \{x \in supp(\mu) \mid \forall x' \in B^o(x, r), \\
&\quad \forall x'' \in B(x', r_p(x')), \eta(x'') > 1/2 + \Delta\} \\
\mathcal{X}^-_{r,\Delta,p} &= \{x \in supp(\mu) \mid \forall x' \in B^o(x, r), \\
&\quad \forall x'' \in B(x', r_p(x')), \eta(x'') < 1/2 - \Delta\}
\end{aligned}$$

What is the significance of these interiors? Let x' be an instance such that all $x'' \in B(x', r_p(x'))$ have $\eta(x'') > 1/2 + \Delta$. If $p \approx \frac{k}{n}$, then the k points x'' closest to x' have $\eta(x'') > 1/2 + \Delta$. Provided the average of the labels of these points is close to expectation, which happens when k is large relative to $1/\Delta$, k-nearest neighbor outputs label 1 on x'. When x is in the r-robust (p, Δ)-strict interior region $\mathcal{X}^+_{r,\Delta,p}$, this is true for all x' within distance r of x, which means that k-nearest neighbors will be robust at x. Thus, the r-robust (p, Δ)-strict interior is the region where we natually expect k-nearest neighbor to have robustness radius r, when k is large relative to $\frac{1}{\Delta}$ and $p \approx \frac{k}{n}$.

Readers familiar with [CD14] will observe that the set of all x' for which $\forall x'' \in B(x', r_p(x')), \eta(x'') > 1/2 + \Delta$ forms a stricter version of the (p, Δ)-interiors of the 1 region that was defined in this work; these x' also represent the region where k-nearest

neighbors are accurate when $k \approx \max(np, 1/\Delta^2)$. The r-robust (p, Δ)-strict interior is thus a somewhat stricter and more robust version of this definition.

Main Results

We begin by characterizing where the Bayes Optimal classifier is robust.

Theorem 3.3.2. *The Bayes Optimal classifier has robustness radius r at $x \in \mathcal{X}^+_{r,0,0} \cup \mathcal{X}^-_{r,0,0}$. Moreover, its astuteness is $\mathbb{E}[\eta(x)\mathbf{1}(x \in \mathcal{X}^+_{r,0,0})] + \mathbb{E}[(1-\eta(x))\mathbf{1}(x \in \mathcal{X}^-_{r,0,0})]$.*

The proof is in the Appendix, along with analogous results for astuteness. The following theorem, along with a similar result for astuteness, proved in the Appendix, characterizes robustness in the large k regime.

Theorem 3.3.3. *For any n, pick a δ and a $\Delta_n \to 0$. There exist constant C_1 and C_2 such that if $k_n \geq C_1 \frac{\sqrt{dn \log n + n \log(1/\delta_n)}}{\Delta_n}$, and $p_n \geq \frac{k_n}{n}(1 + C_2\sqrt{\frac{d \log n + \log(1/\delta)}{k_n}})$, then, with probability $\geq 1 - 3\delta$, k_n-NN has robustness radius r in $x \in \mathcal{X}^+_{r,\Delta_n,p_n} \cup \mathcal{X}^-_{r,\Delta_n,p_n}$.*

Remarks. Some remarks are in order. First, observe that as $n \to \infty$, Δ_n and p_n tend to 0; thus, provided certain continuity conditions hold, $\mathcal{X}^+_{r,\Delta_n,p_n} \cup \mathcal{X}^-_{r,\Delta_n,p_n}$ approaches $\mathcal{X}^+_{r,0,0} \cup \mathcal{X}^-_{r,0,0}$, the robustness region of the Bayes Optimal classifier.

Second, observe that as r-robust strict interiors extend the definition of interiors in [CD14], Theorem 3.3.3 is a robustness analogue of Theorem 5 in this work. Unlike the latter, Theorem 3.3.3 has a more stringent requirement on k. Whether this is necessary is left as an open question for future work.

3.4 A Robust 1-NN Algorithm

Section 3.3 shows that nearest neighbors is robust for k as large as $\Omega(\sqrt{dn \log n})$. However, this k is too high to use in practice – high values of k require even higher sample

sizes [CD14], and lead to higher running times. Thus a natural question is whether we can find a more robust version of the algorithm for smaller k. In this section, we provide a more robust version of 1-nearest neighbors, and analytically demonstrate its robustness.

Our algorithm is motivated by the observation that 1-nearest neighbor is robust when oppositely labeled points are far apart, and when test points lie close to training data. Most training datasets however contain nearby points that are oppositely labeled; thus, we propose to remove a subset of training points to enforce this property.

Which points should we remove? A plausible approach is to keep the largest subset where oppositely labeled points are far apart; however, this subset has poor stability properties even for large n. Therefore, we propose to keep all points x such that: (a) we are highly confident about the label of x and its nearby points and (b) all points close to x have the same label. Given that all such x are kept, we remove as few points as possible, and execute nearest neighbors on the remaining dataset.

The following definition characterizes data where oppositely labeled points are far apart.

Definition 3.4.1 (r-separated set)**.** *A set $A = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ of labeled examples is said to be r-separated if for all pairs $(x_i, y_i), (x_j, y_j) \in A$, $\|x_i - x_j\| \leq r$ implies $y_i = y_j$.*

The full algorithm is described in Algorithm 1 and Algorithm 2. Given confidence parameters Δ and δ, Algorithm 2 returns a 0/1 label when this label agrees with the average of k_n points closest to x; otherwise, it returns $\perp$. k_n is chosen such that with probability $\geq 1 - \delta$, the empirical majority of k_n labels agrees with the majority in expectation, provided the latter is at least Δ away from $\frac{1}{2}$.

Algorithm 2 is used to determine whether an x_i should be kept. Let $f(x_i)$ be the output of Algorithm 2 on x_i. If $y_i = f(x_i)$ and if for all $x_j \in B(x_i, r)$, $f(x_i) = f(x_j) = y_i$, then we mark x_i as red. Finally, we compute the largest r-separated subset of the training data that includes all the red points; this reduces to a constrained matching problem as

in [KW15]. The resulting set, returned by Algorithm 1, is our new training set. We observe that this set is r-separated from Lemma A.2.2 in the Appendix, and thus oppositely labeled points are far apart. Moreover, we keep all (x_i, y_i) when we are confident about the label of x_i and its nearby points. Observe that our final procedure is a 1-NN algorithm, even though k_n neighbors are used to determine if a point should be retained in the training set.

Algorithm 1 Robust-1NN(S_n, r, Δ, δ, x)

for $(x_i, y_i) \in S_n$ **do**
  $f(x_i) =$ Confident-Label$(S_n, \Delta, \delta, x_i)$
end for
$S_{RED} = \emptyset$
for $(x_i, y_i) \in S_n$ **do**
  if $f(x_i) = y_i$ and $f(x_i) = f(x_j)$ for all x_j such that $\|x_i - x_j\| \leq r$ and $(x_j, y_j) \in S_n$ **then**
    $S_{RED} = S_{RED} \bigcup \{(x_i, y_i)\}$
  end if
end for
Let S' be the largest $2r$-separated subset of S_n that contains all points in S_{RED}.
return new training set S'

Algorithm 2 Confident-Label(S_n, Δ, δ, x)

$k_n = 3\log(2n/\delta)/\Delta^2$
$\bar{y} = (1/k_n)\sum_{i=1}^{k_n} Y^{(i)}(x)$
if $\bar{y} \in [\frac{1}{2} - \Delta, \frac{1}{2} + \Delta]$ **then**
  return $\perp$
else
  return $\frac{1}{2}sgn(\bar{y} - \frac{1}{2}) + \frac{1}{2}$
end if

3.4.1 Performance Guarantees

The following theorem establishes performance guarantees for Algorithm 1.

Theorem 3.4.2. *Pick a Δ_n and δ, and set $k_n = 3\log(2n/\delta)/\Delta_n^2$. Pick a* margin parameter *τ. Then, there exist constants C and C_0 such that the following hold. If we set $p_n =$*

$\frac{k_n}{n}(1 + C\sqrt{\frac{d \log n + \log(1/\delta)}{k_n}})$, *and define the set:*

$$X_R = \left\{ x \middle| x \in \mathcal{X}^+_{r+\tau,\Delta_n,p_n} \cup \mathcal{X}^-_{r+\tau,\Delta_n,p_n}, \right.$$
$$\left. \mu(B(x,\tau)) \geq \frac{2C_0}{n}(d \log n + \log(1/\delta)) \right\}$$

Then, with probability $\geq 1 - 2\delta$ *over the training set, Algorithm 1 run with parameters* r, Δ_n *and* δ *has robustness radius at least* $r - 2\tau$ *on* X_R.

Remarks. The proof is in the Appendix, along with an analogous result for astuteness. Observe that X_R is roughly the *high density subset* of the $(r + \tau)$-robust strict interior $\mathcal{X}^+_{r+\tau,\Delta_n,p_n} \cup \mathcal{X}^-_{r+\tau,\Delta_n,p_n}$. Since $\eta(x)$ is constrained to be greater than $\frac{1}{2} + \Delta_n$ or less than $\frac{1}{2} - \Delta_n$ in this region, as opposed to 0 or 1, this is an improvement over standard nearest neighbors when the data distribution has a large high density region that intersects with the interiors.

A second observation is that as τ is an arbitrary constant, we can set to it be quite small and still satisfy the condition on $\mu(B(x,\tau))$ for a large fraction of x's when n is very large. This means that in the large sample limit, $r - 2\tau$ may be close to r and X_R may be close to the high density subset of $\mathcal{X}^+_{r,\Delta_n,p_n} \cup \mathcal{X}^-_{r,\Delta_n,p_n}$ for a lot of smooth distributions.

3.5 Experiments

The results in Section 3.4 assume large sample limits. Thus, a natural question is how well Algorithm 1 performs with more reasonable amounts of training data. We now empirically investigate this question.

Since there are no general methods that certify robustness at an input, we assess robustness by measuring how our algorithm performs against a suite of standard attack

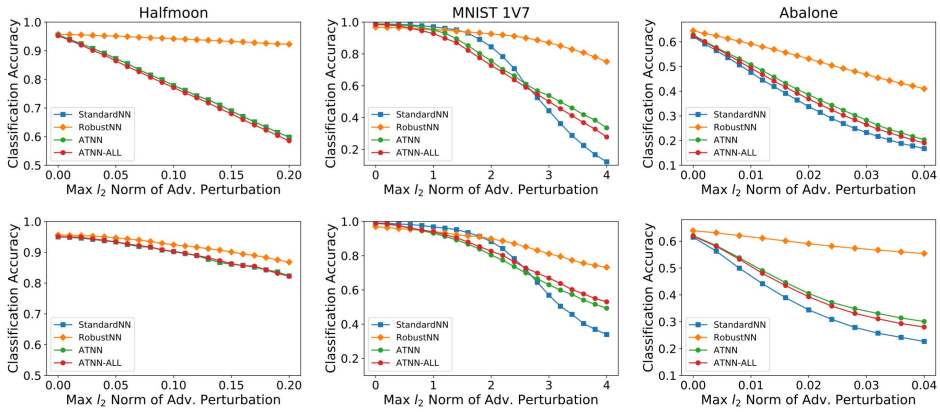

Figure. 3.1: **White Box Attacks:** Plot of classification accuracy on adversarial examples v.s. attack radius. *Top row:* Direct Attack. *Bottom row:* Kernel Substitute Attack. *Left to right:* 1) Halfmoon, 2) MNIST 1v 7 and 3) Abalone.

methods. Specifically, we consider the following questions:

1. How does our algorithm perform against popular white box and black box attacks compared with standard baselines?

2. How is performance affected when we change the training set size relative to the data dimension?

These questions are considered in the context of three datasets with varying training set sizes relative to the dimension, as well as two standard white box attacks and black box attacks with two kinds of substitute classifiers.

3.5.1 Methodology

Data

We use three datasets – Halfmoon, MNIST 1v7 and Abalone – with differing data sizes relative to dimension. Halfmoon is a popular 2-dimensional synthetic data set for non-linear classification. We use a training set of size 2000 and a test set of size 1000 generated with standard deviation $\sigma = 0.2$. The MNIST 1v7 data set is a subset of the

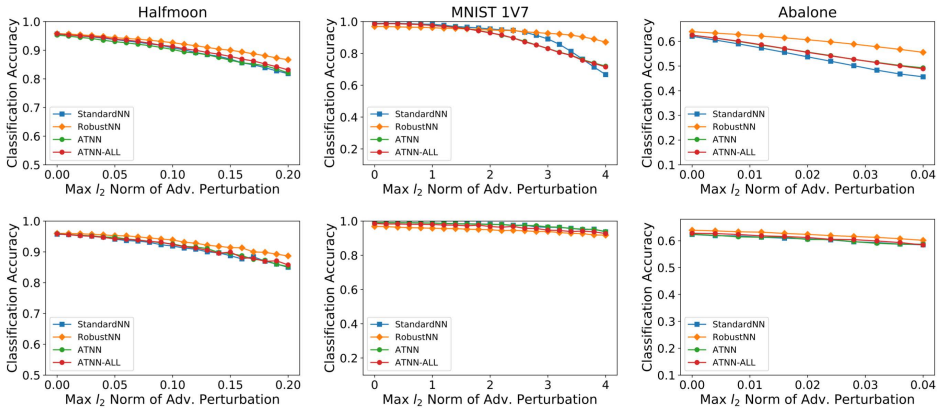

Figure. 3.2: **Black Box Attacks:** Plot of classification accuracy on adversarial examples v.s. attack radius. *Top to Bottom:* 1) kernel substitute, 2) neural net substitute. *Left to right:* 1) Halfmoon, 2) MNIST 1 v.s. 7 and 3) Abalone.

784-dimensional MNIST data. For training, we use 1000 images each of Digit 1 and 7, and for test, 500 images of each digit. Finally, for the Abalone dataset [Lic13], our classification task is to distinguish whether an abalone is older than 12.5 years based on 7 physical measurements. For training, we use 500 and for test, 100 samples. In addition, a validation set with the same size as the test set is generated for each experiment for parameter tuning.

Baselines

We compare Algorithm 1, denoted by RobustNN, against three baselines. The first is the standard 1-nearest neighbor algorithm, denoted by StandardNN. We use two forms of adversarially-trained nearest neighbors - ATNN and ATNN-all. Let S be the training set used by standard nearest neighbors. In ATNN, we augment S by creating, for each $(x, y) \in S$, an adversarial example x_{adv} using the attack method in the experiment, and adding (x_{adv}, y). The ATNN classifier is 1-nearest neighbor on this augmented data. In ATNN-all, for each $(x, y) \in S$, we create adversarial examples using *all* the attack methods in the experiment, and add them all to S. ATNN-all is the nearest neighbor classifier on this augmented data. For example, for white box Direct Attacks in Section 3.5.2, ATNN includes adversarial examples generated by the Direct Attack, and ATNN-all includes

adversarial examples generated by both Direct and Kernel Substitute Attacks.

Observe that all algorithms except StandardNN have parameters to tune. RobustNN has three input parameters – Δ, δ and a defense radius r which is an approximation to the robustness radius. For simplicity, we set $\Delta = 0.45$, $\delta = 0.1$ and tune r on the validation set; this can be viewed as tuning the parameter τ in Theorem 3.4.2. For ATNN and ATNN-all, the methods that generate the augmenting adversarial examples need a perturbation magnitude r; we call this the *defense radius*. To be fair to all algorithms, we tune the defense radius for each. We consider the adversary with the highest attack perturbation magnitude in the experiment, and select the defense radius that yields the highest validation accuracy against this adversary.

3.5.2 White-box Attacks and Results

To evaluate the robustness of Algorithm 1, we use two standard classes of attacks – white box and black box. For white-box attacks, the adversary knows all details about the classifier under attack, including its training data, the training algorithm and any hyperparameters.

Attack Methods

We consider two white-box attacks – direct attack [ABE+16] and Kernel Substitute Attack [PMG16].

Direct Attack. This attack takes as input a test example x, an attack radius r, and a training dataset S (which may be an augmented or reduced dataset). It finds an $x' \in S$ that is closest to x but has a different label, and returns the adversarial example $x_{adv} = x + r\frac{x-x'}{||x-x'||_2}$.

Kernel Substitute Attack. This method attacks a substitute kernel classifier trained on the *same training set*. For a test input $\vec{x}$, a set of training points Z with one-hot

labels Y, a kernel classifier f predicts the class probability as:

$$f : \vec{x} \to \frac{\left[e^{-||\vec{x}-\vec{z}||_2^2/c}\right]_{\vec{z}\in X}}{\sum_{\vec{z}\in X} e^{-||\vec{x}-\vec{z}||_2^2/c}} \cdot Y$$

The adversary trains a kernel classifier on the training set of the corresponding nearest neighbors, and then generates adversarial examples against this kernel classifier. The advantage is that the prediction of the kernel classifier is differentiable, which allows the use of standard gradient-based attack methods. For our experiments, we use the popular fast-gradient-sign method (FSGM). The parameter c is tuned to yield the most effective attack, and is set to 0.1 for Halfmoon and MNIST, and 0.01 for Abalone.

Results

Figure 3.1 shows the results. We see that RobustNN outperforms all baselines for Halfmoon and Abalone for all attack radii. For MNIST, for low attack radii, RobustNN's classification accuracy is slightly lower than the others, while it outperforms the others for large attack radii. Additionally, as is to be expected, the Direct Attack results in lower general accuracy than the Kernel Substitute Attack.

These results suggest that our algorithm mostly outperforms the baselines StandardNN, ATNN and ATNN-all. As predicted by theory, the performance gain is higher when the training set size is large relative to the dimension – which is the setting where nearest neighbors work well in general. It has superior performance for Halfmoon and Abalone, where the training set size is large to medium relative to dimension. In contrast, in the sparse dataset MNIST, our algorithm has slightly lower classification accuracy for small attack radii, and higher otherwise.

3.5.3 Black-box Attacks and Results

[PMG+17] has observed that some defense methods that work by masking gradients remain highly amenable to *black box attacks*. In this attack, the adversary is unaware of the target classifier's nature, parameters or training data, but has access to a seed dataset drawn from the same distribution which they use to train and attack a substitute classifier. To establish robustness properties of Algorithm 1, we therefore validate it against black box attacks based on two types of substitute classifiers.

Attack Methods

We use two types of substitute classifiers – kernel classifiers and neural networks. The adversary trains the substitute classifier using the method of [PMG+17] and uses the adversarial examples against the substitute to attack the target classifier.

Kernel Classifier. The kernel classifier substitute is the same as the one in Section 3.5.2, but trained using the seed data and the method of [PMG+17].

Neural Networks. The neural network for MNIST is the ConvNet in [PCG+17]'s tutorial. For Halfmoon and Abalone, the network is a multi-layer perceptron with 2 hidden layers.

Procedure. To train the substitute classifier, the adversary uses the method of [PMG16] to augment the seed data for two rounds; labels are obtained by querying the target classifier. Adversarial examples against the substitutes are created by FGSM, following [PMG16]. As a sanity check, we verify the performance of the substitute classifiers on the original training and test sets. Details are in Table A.1 in the Appendix. Sanity checks on the performance of the substitute classifiers are presented in Table A.1 in the Appendix.

Results

Figure 3.2 shows the results. For all algorithms, black box attacks are less effective than white box, which corroborates the results of [PMG16], who observed that black-box attacks are less successful against nearest neighbors. We also find that the kernel substitute attack is more effective than the neural network substitute, which is expected as kernel classifiers have similar structure to nearest neighbors. Finally, for Halfmoon and Abalone, our algorithm outperforms the baselines for both attacks; however, for MNIST neural network substitute, our algorithm does not perform as well for small attack radii. This again confirms the theoretical predictions that our algorithm's performance is better when the training set is large relative to the data dimension – the setting in which nearest neighbors work well in general.

3.5.4 Discussion

The results show that our algorithm performs either better than or about the same as standard baselines against popular white box and black box attacks. As expected from our theoretical results, it performs better for denser datasets which have high or medium amounts of training data relative to the dimension, and either slightly worse or better for sparser datasets, depending on the attack radius. Since non-parametric methods such as nearest neighbors are mostly used for dense data, this suggests that our algorithm has good robustness properties even with reasonable amounts of training data.

3.6 Conclusion

We introduce a novel theoretical framework for learning robust to adversarial examples, and introduce notions of distributional and finite-sample robustness. We use these notions to analyze a non-parametric classifier, k-nearest neighbors, and introduce a

novel modified 1-nearest neighbor algorithm that has good robustness properties in the large sample limit. Experiments show that these properties are still retained for reasonable data sizes.

3.7 Acknowledgement

Chapter 3 is based on the material in International Conference of Machine Learning 2018 (Yizhen Wang, Somesh Jha and Kamalika Chaudhuri,"Analyzing the Robustness of Nearest Neighbors to Adversarial Examples"). The dissertation author is the primary investigator and co-author of this material.

Chapter 4

Robust Machine Learning-Based Malware Detection Against Program Transformation

4.1 Introduction

In this chapter, we consider the adversarial attack in malware detection task, in which the attacker can change the syntax of the malicious software to evade detection while still maintain the original malicious function. We generalize potential adversarial transformation as logical relations, and evaluate two defense approaches – program normalization and robust optimization – in both theory and practice. The program normalization approach is robust by design.

Malware detection is a challenging problem in computer security and past approaches based on pattern matching, although widely deployed and efficient, are reactive as they detect only past threats. The success of machine learning in other data-driven classification tasks in areas such as computer vision and natural language processing has drawn the

attention of malware-detection researchers, who currently focus on using machine learning to identify previously unseen malware samples [PSS+15, JSD+17, OPL13].

While machine learning-based malware detection has shown strong predictive power, it has been shown that machine learning is open to carefully designed threats, as a number of attacks showed that robustness of learned classifiers is not gained by default [MMS+18, BR17, CW17]. Past work in adversarial machine learning observed that a variety of attacks against malware detectors can be constructed, from poisoning of training data sets (consisting of malware samples that are controlled by the adversary) [MZ15, BNL12] to evasion via controlled misclassification [BCM+13, PMJ+16, SZS+13]. A solution that addresses such attacks is to (re)train the machine-learning model, for example through an adversarial-training procedure [MMS+18], in order to refine the detection boundary and produce a new model with higher classification accuracy under adversarial attack.

Similar to natural language processing, malware detection operates in a discrete feature domain. A feature is often a boolean variable, representing the presence of a certain characteristic in the input program file or the occurrence of a certain behavior in its execution. Typical examples include whether the program opens a certain file and whether the program calls a certain API such as `memcpy`. In this discrete setting the threat model in recent robust malware detection research is that of feature addition, under the assumption that it is easier to add a new, extraneous feature to the program than to remove a necessary one. This addition threat model is assumed by recent malware detection attacks [GPM+16, GPM+17] and defenses [ITAW18, ADHHO18].

In this chapter, we challenge this commonly used threat model. We present new attacks that can remove features from the input program and corresponding new defense techniques. Our work is motivated by *program obfuscation*, a program transformation that preserve semantics while changing syntactic structure to any degree desired. Applied to the malware domain, obfuscation allows an adversary to create an unlimited number of

new malicious samples. Semantics-preserving program transformations can be as simple as replacing an API with another, such as replacing a call to `printf` library function with a call to the `cout` library function. Following this approach, our new attacks are based on replacing APIs with other equivalent APIs and are particularly potent against malware-detection techniques that use as features the presence of APIs.

On the defense side, we abstract the impact of both addition and substitution (and any other as program transformations) on the feature vector as a *relations* between the features. For example, there is an equivalence relation between `printf` and `cout` and intuitively a malware should be still classified as a malware after progrma transformations. This abstraction step allows us to reduce the problem of constructing robust malware detectors to that of learning a class while respecting a relation. However, machine learning does not necessarily learn classifiers that respect a relation by default. Therefore, we believe it is critical to develop machine learning techniques that *respect a relation*.

When considering the task of learning a classifier that respects a relation, a natural question is whether the classifier can automatically learn the relation from the training data, while under adversarial settings. This is the basic idea behind using adversarial training [MMS+18]. We evaluate our substitution attacks against a state-of-the-art robust malware detection technique [ADHHO18] and observe a 82.8% False Negative Rate (FNR) for robust classifiers, whose FNR is under 10% against addition attacks. These results show that substitution attacks pose real threats to robust malware detection and future research on robust malware detection should also question the practicality of the addition threat model.

Another approach to learning a classifier that respects a relation is to use *normalization*, a technique introduced in network security and extended to programs when applied to malware detection. Program normalization is used in practice as a way to reverse the effects of obfuscation, by transforming a program to a semantically equivalent one that

is simpler and thus easier to analyze and classify. A normalization technique induces an equivalence relation over the set of all programs, with equivalence classes each having one unique representative sample (a normalized program). In turn this can be used in the malware detection task by applying normalization as a preprocessing step during both training and inference and then using a machine-learning technique that is not necessarily enhanced via adversarial training. We call this approach *normalize-and-predict*. It is an open question whether normalization, initially used to help rule-based detection systems, is useful or necessary for machine learning.

We compare adversarial learning with program normalization from both theoretical and empirical perspectives. Our evaluation shows that adversarial training can achieve a 12.7% FNR, but is not as effective against substitution attacks as normalize-and-predict, with a 7.4% FNR. It is surprising to us that adversarial training can achieve any reasonable success against substitution attacks despite its probablistic nature. Therefore, we attempt to understand how the adversarially trained models respect the substitution relation and introduce a formalism for learning a relation and for analyzing and comparing adversarial training and normalize-and-predict.

Our paper makes the following contributions:

- We provide a framework to reason about robustness of malware detectors in the presence of program transformations. Fundamentally this reduces to the task of constructing classifiers that respect a relation, such as program equivalence under a set of code transformations (e.g., code addition, code substitution). (Sections 4.2)

- We present two novel attacks based on an API substitution relation and show that these attacks are effective against the state-of-the-art robust malware detection classifiers. (section 4.5)

- We show that normalize-and-predict and adversarial training are two viable approach

for learning classifiers that respect a relation. We show that these two approaches optimize for different objectives, that normalize-and-predict can enable robust linear classifiers whereas adversarial training may fail to do so, and empirically evaluate how they perform in practice. (Sections 4.4 and 6.5)

4.2 The Setting

4.2.1 Learning via Empirical Risk Minimization

We assume a data distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the space of samples and $\mathcal{Y}$ is the space of labels. Let $\mathcal{D}_{\mathcal{X}}$ be the marginal distribution over $\mathcal{X}$ induced by $\mathcal{D}$.[1] In the *empirical risk minimization (ERM)* framework we wish to solve the following optimization problem:

$$\min_{w \in \mathcal{H}} \; E_{(\mathbf{x},y)\sim\mathcal{D}}\, l(w, \mathbf{x}, y)$$

In the equation given above $\mathcal{H}$ is the hypothesis space and l is the loss function. We will denote vectors in boldface (e.g. $\mathbf{x}$, $\mathbf{y}$). Since the distribution is usually unknown, a learner solves the following problem over a data set $S = \{(\mathbf{x}_1, y_1), \cdots .(\mathbf{x}_n, y_n)\}$

$$\min_{w \in \mathcal{H}} \; \frac{1}{n} \sum_{i=1}^{n} l(w, \mathbf{x}_i, y_i)$$

Once we have solved the optimization problem given above, we obtain a w^* which yields a classifier $F : \mathcal{X} \to \mathcal{Y}$ (the classifier is usually parameterized by w^*, but we will omit it for brevity).

For the rest of the chapter, we will focus on the special case when $\mathcal{X} = \{0,1\}^n$ (the sample space is n-dimensional Boolean vectors) and $\mathcal{Y} = \{0,1\}$ (we consider binary classification). A typical example of this special case is malware detection where each

[1] The measure of set $Z \subseteq \mathcal{X}$ in distribution $\mathcal{D}_{\mathcal{X}}$ is the measure of the set $Z \times \mathcal{Y}$ in distribution $\mathcal{D}$.

feature is Boolean value representing whether a certain API call is present in the program, and the classification result is binary: malicious or benign. However, our remarks readily transfer to the general problem. A vector $\mathbf{x} \in \{0,1\}^n$ will be denoted in boldface. Note that for our special case our classifier $F : \{0,1\}^n \to \{0,1\}$ can be regarded as a Boolean function, so we will sometimes refer to our classifier as a Boolean function.

4.2.2 Relation, and Function that Respects Relation

A *relation* $\mathcal{R}$ is a subset of $\{0,1\}^n \times \{0,1\}^n$. We will write $\mathbf{x} \to_{\mathcal{R}} \mathbf{y}$ iff $(\mathbf{x}, \mathbf{y}) \in \mathcal{R}$. We will write $\mathbf{x} \to^*_{\mathcal{R}} \mathbf{y}$ iff $\mathbf{x} = \mathbf{y}$ or there exists $\mathbf{y}_0, \mathbf{y}_1, \cdots, \mathbf{y}_k$ $(k > 0)$ such that $\mathbf{x} = \mathbf{y}_0$, $\mathbf{y}_i \to_{\mathcal{R}} \mathbf{y}_{i+1}$ $(0 \leq i < k)$ and $\mathbf{y}_k = \mathbf{y}$. In other words, $\to^*_{\mathcal{R}}$ is the reflexive-transitive closure of $\to_{\mathcal{R}}$.

Example. To make things concrete, we will use malware detection as our running example. Let $\mathcal{M}$ be the space of malicious programs and $\mathcal{Z} : \mathcal{M} \to \{0,1\}^n$ be a "feature extractor" (i.e. $\mathcal{Z}(m)_i = 1$ means that the i-th system call is present in the malware m). From now, we will equate malware with its "feature vector" representation in the Boolean space $\{0,1\}^n$.

Let $f^\star : \{0,1\}^n \to \{0,1\}$ be a Boolean function that corresponds to "ground truth" (the reader can use the following intuition: $f^\star(\mathbf{x}) = 1$ means that $\mathbf{x}$ corresponds to a malware and $f^\star(\mathbf{x}) = 0$ means that $\mathbf{x}$ corresponds to a benign program). Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function that is supposed to approximate $f^\star$ and is the output of a learning algorithm (e.g. an algorithm for learning decision trees to classify executables as malicious or benign).

We say that a Boolean function g *respects* the relation $\mathcal{R}$ iff for all $\mathbf{x}$ and $\mathbf{y}$ in $\{0,1\}^n$ such that $\mathbf{x} \to^*_{\mathcal{R}} \mathbf{y}$ and $g(\mathbf{x}) = 1$ implies that $g(\mathbf{y}) = 1$. In other words, the relation $\mathcal{R}$ "preserves maliciousness" (if $\mathbf{x}$ is classified as malware, then a transformed malware $\mathbf{y}$ is also classified as malware). We assume that $f^\star$ respects the relation $\mathcal{R}$. Recall that f is

the approximate or learned version of $f^\star$. Ideally, we also want f to respect $\mathcal{R}$. However, we can also define a probabilistic version of the "respects condition" that might be more appropriate for machine-learning techniques. Let $A(\mathcal{R}, f, \mathbf{x}, \mathbf{y})$ be the following predicate:

$$(\mathbf{x} \rightarrow^*_{\mathcal{R}} \mathbf{y} \ \wedge f(\mathbf{x}) = 1) \Rightarrow f(\mathbf{y}) = 1$$

A "probabilistic" version of the above definition is defined as the following metric (where $\mathcal{D}$ is the data-generating distribution over $\{0,1\}^n$):

$$\mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\sim(\mathcal{D}_\mathcal{X}\times\mathcal{D}_\mathcal{Y})} \mathbb{1}_{A(\mathcal{R},f,\mathbf{x},\mathbf{y})}$$

4.2.3 Some Example Relations

Next we formalize some of the common relation that are specific to our context– malware detection. Let $x_1, \cdots, x_n$ be Boolean variables. A literal is x_i or $\neg x_i$. A clause is a conjunction of literals (e.g. $x_2 \wedge x_4 \wedge \neg x_6$) A 2-tuple of clauses (C, C') defines a relation as follows: $(\mathbf{x}, \mathbf{y})$ is in the relation iff $\mathbf{x} \models C$ and $\mathbf{y} \models C'$.[2] A relation $\mathcal{R}$ can then be defined as a set of 2-tuples of clauses $\{(C_1, C'_1), \cdots, (C_k, C'_k)\}$ with the natural interpretation (i.e. $(\mathbf{x}, \mathbf{y}) \in \mathcal{R}$ iff there exists $i \in \{1, \cdots, k\}$ such that $\mathbf{x} \models C_i$ and $\mathbf{x} \models C'_i$).

A *monotone relation* R_m can be defines as follows: $\{(\neg x_1, x_1), \cdots, (\neg x_n, x_n)\}$. Recall that a monotone relation corresponds to a malware author adding unused system calls. A *substitution relation* R_s can be defines as a set of clauses, which have the following form $(x_{i_1} \wedge \cdots \wedge x_{i_k}, x_{j_1} \wedge \cdots \wedge x_{j_m})$, where $\{i_1, \cdots i_k\} \bigcap \{j_1, \cdots, j_m\} = \emptyset$. Informally, the relation $(x_{i_1} \wedge \cdots \wedge x_{i_k}, x_{j_1} \wedge \cdots \wedge x_{j_m})$ represents features $\{i_1, \cdots i_k\}$ being substituted by $\{j_1, \cdots, j_m\}$ (this corresponds to an adversary replacing a sequence of system-calls by another sequence of systems-calls). From now on, we we will assume that our relation

[2] $\mathbf{x} \models C$ iff the vector corresponding to $\mathbf{x}$ is a satisfying assignment for C.

is $R_m \cup R_s$ (i.e. we allow monotone bit flipping and substitutions). If we only consider monotone relation, we can set $R_s = \emptyset$.

Example. *Simple substitution.* We again consider the example of malware detection. Consider $\mathbf{x} \in \{0,1\}^n$, where $\mathbf{x}_i = 1$ means that the corresponding sample contains the i-th system call. Suppose we are given a partition $\{P_1, \cdots, P_k\}$ of the set $\{1, \cdots, n\}$, where if i and j belong to the same partition P_r, then they can be substituted with one another (e.g. the system calls corresponding to i and j are equivalent). This can be easily put in the formulation we described before, and the partition corresponds to a relation in a natural manner as follows: for each pair of indices (i, j) that are in the same partition we have two pairs of clauses (x_i, x_j) and (x_j, x_i). As we will describe in 4.6.2, Windows contains a wide range of equivalent APIs for attackers to apply simple substitution in a large substitution space.

4.3 Normalize-and-Predict and Robust Optimization – Two Approaches of Learning Classifiers that Respect a Relation

Let $\mathcal{F}$ be the hypothesis space of Boolean functions. We want to learn a Boolean function that respects a relation $\mathcal{R}$. Therefore, we want to constrain the space of Boolean functions $\mathcal{F}$. In other words, we want to find the maximal set $\mathcal{F}_\mathcal{R} \subseteq \mathcal{F}$ such that all $f \in \mathcal{F}_\mathcal{R}$ respect $\mathcal{R}$. If we have a representation of $\mathcal{F}_\mathcal{R}$ we can define our the ERM problem as:

$$\min_{f \in \mathcal{F}_\mathcal{R}} \; E_{(\mathbf{x},y)\sim\mathcal{D}}\, l(w, \mathbf{x}, y)$$

In several cases, $\mathcal{F}_{\mathcal{R}}$ is hard to characterize. Therefore, we outline two approaches to address this difficulty – one that is based on adversarial training and another which will call *normalize-and-predict (NaP)*.

Robust Optimization. In such cases, we can use *robust optimization*, which solves the following optimization problem:

$$\min_{f \in \mathcal{F}} E_{(\mathbf{x},y)\sim\mathcal{D}} \max_{\mathbf{z}\in \text{post}_{\mathcal{R}}(\mathbf{x})} l(w, \mathbf{z}, y)$$

In the equation given above, $\text{post}_{\mathcal{R}}(\mathbf{x})$ denotes the following set:

$$\{\mathbf{y} \mid \mathbf{x} \rightarrow^*_{\mathcal{R}} \mathbf{y}\}$$

In other words, $\mathbf{y}$ can be obtained from $\mathbf{x}$ using 0 or more rewrites according to the relation $\mathcal{R}$. The robust optimization problem is usually solved using *adversarial training*. Adversarial training works as follows: let $\mathcal{A}$ be an attack algorithm which given an $\mathbf{x}$ attempts to find $\mathbf{z}$ according to the objective $\max_{\mathbf{z}\in\text{post}_{\mathcal{R}}(\mathbf{x})} l(w, \mathbf{z}, y)$ (an attacker tries to find a "transformed" example that maximizes the loss). Note that $\mathcal{A}$ might not achieve the objective exactly because the inner maximization problem can be hard to solve. Given $\mathcal{A}$, each data $\mathbf{x}_i$ is replaced by $\mathcal{A}(\mathbf{x}_i)$, or each sample is replaced by the corresponding adversarial example. With adversarial training a learner solves the following problem over a data set $S = \{(\mathbf{x}_1, y_1), \cdots .(\mathbf{x}_n, y_n)\}$

$$\min_{w\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} l(w, \mathcal{A}(\mathbf{x}_i), y_i)$$

Normalize-and-Predict (NaP). Let $\preceq$ be a preorder on the set $\{0,1\}^n$.[3] If $a \preceq b$ and $a \neq b$, we will write it as $a \prec b$. A metric μ over $\{0,1\}^n$ induces a preorder corresponding to the

[3]Recall that a preorder satisfies two properties: reflexivity and transitivity.

value of the metric ($a \preceq b$) iff $\mu(a) \leq \mu(b)$. We say that a relation $\mathcal{R}$ *respects* the preorder $\preceq$ iff $(\mathbf{x}, \mathbf{y}) \in \mathcal{R}$ implies that $\mathbf{y} \preceq \mathbf{x}$.

We also define an undirected graph G, in which

- the nodes are all lattices in $\{0,1\}^n$, and
- an edge $(\mathbf{x}, \mathbf{z})$ exists *iff* $\mathbf{x} \rightarrow_{\mathcal{R}}^{*} \mathbf{z}$ or $\mathbf{z} \rightarrow_{\mathcal{R}}^{*} \mathbf{x}$.

For a vector x, we use $S_{\mathcal{R}}(\mathbf{x})$ to denote the connected component in G that contains $\mathbf{x}$, i.e.

$$S_{\mathcal{R}}(\mathbf{x}) = \{\mathbf{z} \,|\text{a path exists between } \mathbf{x} \text{ and } \mathbf{z} \text{ in } G\}.$$

Given $\mathbf{x}$ and $\mathcal{R}$, we assume that it has a normal form $N(\mathbf{x}, \mathcal{R})$, which satisfies the following properties:

- $N(\mathbf{x}, \mathcal{R}) \in S_{\mathcal{R}}(\mathbf{x})$
- There does not exist a $\mathbf{z}$ such that $\mathbf{z} \in S_{\mathcal{R}}(\mathbf{x})$ and $\mathbf{z} \prec N(\mathbf{x}, \mathcal{R})$

Moreover, we assume that given $\mathbf{x}$ and $\mathcal{R}$ there is an effective algorithm to compute $N(\mathbf{x}, \mathcal{R})$. We call $N(\mathbf{x}, \mathcal{R})$ the *normal form* of $\mathbf{x}$. *NaP* framework works as follows:

- *Training:* Essentially each data item is normalized before training. With adversarial training a learner solves the following problem over a data set $S = \{(\mathbf{x}_1, y_1), \cdots .(\mathbf{x}_n, y_n)\}$

$$\min_{w \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} l(w, N(\mathbf{x}_i, \mathcal{R}), y_i)$$

- *Predict:* Let $f : \{0,1\}^n \rightarrow \{0,1\}$ that is constructed after training. Prediction of $\mathbf{x} \in \{0,1\}^n$ is $f(N(\mathbf{x}, \mathcal{R}))$ (we normalize the sample before passing it through the classifier).

Example. We revisit the simple substitution example described earlier in the context of malware detection. Consider $\mathbf{x} \in \{0,1\}^n$, where $\mathbf{x}_i = 1$ means that the corresponding sample contains the i-th system call. Suppose we are given a partition $\{P_1, \cdots, P_k\}$ of the set $\{1, \cdots, n\}$, where if i and j belong to the same partition P_l, then they can be substituted with one another (e.g. the system calls corresponding to i and j have equivalent semantics). In this case the normalize function $N(\mathbf{x}, \mathcal{R})$ is simple and described as follows: We define $N(\mathbf{x}, \mathcal{R})$ as follows:

$$N(\mathbf{x}, \mathcal{R})_j = \begin{cases} \bigvee_{m \in P_l} \mathbf{x}_m & \text{if } j = \min P_l \\ & \text{for some } l \in \{1, \cdots, n\} \\ 0 & \text{otherwise} \end{cases}$$

In the equation given above $\mathbf{x}_m$ denotes the m-th element of the vector $\mathbf{x}$. It is not hard to see that $N(\cdot, \cdot)$ satisfies the properties outlined before.

4.4 Comparison between Normalize-and-Predict and Robust Optimization

Section 4.3 introduces two candidate approaches of learning classifiers respecting relations. Program normalization has been studied and used widely in many program analysis applications [HPK01, CJK+07, CLD11, BRTV16, BPM17, SB16], while robust optimization has been one of the most effective approaches in the machine learning community for learning classifiers robust to adversarial perturbations [MMS+18, CW17]

Which approach is better for robust malware detection? In this section, we compare the two approaches in terms of their optimization objectives, amount of model capacity required and hardness of training. We show that the two approaches give different types of guarantee when they optimize their respective objectives. The *normalize-and-predict*

approach guarantees to learn classifiers that respect the relation. In contrast, robust optimization only approximately respects the the relation and may violate the relation for better classification accuracy in the adversarial setting. While robust optimization in the best case can have higher classification accuracy in the adversarial setting, it also needs to use a more complex model class that allows richer intermediate feature representation, and thus can be harder to train.

4.4.1 Optimization Objectives

The most easily noticed difference between the two approaches is in their objective functions. To best understand the fundamental difference between the approaches, we first look at their objectives on the data distribution instead on a finite data set to avoid potential interference from generalization gaps. We show that the optimal classifiers for these two objectives have different guarantees. In addition, we also show a special case in which the two objectives actually yield the same classifier in terms of the classification outcome.

As introduced in Section 4.3, *normalize-and-predict* has the following objective

$$\min_{f \in \mathcal{F}} \; E_{(\mathbf{x},y)\sim\mathcal{D}} \, l(w, N(\mathbf{x}, \mathcal{R}), y_i) \tag{4.1}$$

over the data distribution, whereas the adversarial training procedure hopes to solve the following optimization problem

$$\min_{f \in \mathcal{F}} \; E_{(\mathbf{x},y)\sim\mathcal{D}} \max_{\mathbf{z} \in \text{post}_{\mathcal{R}}(\mathbf{x})} \; l(w, \mathbf{z}, y). \tag{4.2}$$

Let f_{NaP} and f_{adv} denote the optimal classifier to Objective 4.1 and 4.2 respectively. The following propositions characterize the property that each classifier guarantees.

Proposition 4.4.1. *The classifier f_{NaP} always respects $\mathcal{R}$, i.e. its prediction is always robust under adversarial rewrites on the test input.*

Proposition 4.4.2. *The classifier f_{adv} has the lowest expected loss against the strongest adversary under $\mathcal{R}$. In particular, when ℓ in Objective 4.2 is 0-1 classification loss, f_{adv} has the lowest classification error over the distribution in the adversarial setting.*

The proof to these propositions is straight-forward. The classifier f_{NaP} respects $\mathcal{R}$ because for any input instance x, all vectors in $\text{post}_{\mathcal{R}}(\mathbf{x})$ will be normalized to $N(\mathbf{x}, \mathcal{R})$, which will then have the same classification results. The proposition of f_{adv} is a direct result of its definition.

The subtle difference between the properties of optimal solution suggests that the practitioners should choose the approach carefully based on their needs. If the prediction on malware needs to be absolutely consistent under $\mathcal{R}$, then *normalize-and-predict* is perferable. Otherwise, the adversarial training approach can potentially offer better classification accuracy at the cost of not respecting the relation sometimes.

Despite the difference, there exists a condition under which the two optimal classifiers f_{NaP} and f_{adv} have the same prediction on all input vectors. This scenario is especially interesting because the learner can now get the best from both worlds – a classifier that guarantees to respect relation $\mathcal{R}$ and meanwhile has the highest classification accuracy against the strongest attacker.

Theorem 4.4.3. *Assume for all $\mathbf{x} \in \mathcal{X}$, $\mathbf{z} \in \text{post}_{\mathcal{R}}(\mathbf{x})$ if and only if $\mathbf{x} \in \text{post}_{\mathcal{R}}(\mathbf{z})$, and consider $\mathcal{F}$ to be the set of all possible labeling functions on $\mathcal{X}$. Suppose f_{NaP} and f_{adv} are unique. Then $f_{NaP}(\mathbf{x}) = f_{adv}(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$.*

Proof. Let $\mathcal{C} = \{\text{post}_{\mathcal{R}}(\mathbf{x}) | \forall\, \mathbf{x} \in \mathcal{X}\}$, i.e. each element in $\mathcal{C}$ is the set of all possible vectors of transforming some $\mathbf{x}$ under $\mathcal{R}$. The assumption in Theorem 4.4.3 ensures that elements in $\mathcal{C}$ partition the entire input space $\mathcal{X}$.

For each $C(\mathbf{x}) = \text{post}_{\mathcal{R}}(\mathbf{x}) \in \mathcal{C}$, we know that

$$f_{NaP}(\mathbf{z}) = \mathbb{1}\left[\Pr(y = 1 | \, \mathbf{z} \in C(\mathbf{x})) > 1/2\right]$$

, which is the Bayes' optimal classifier and is the same for all $\mathbf{z} \in C(\mathbf{x})$. It remains to show that $f_{adv}(\mathbf{z}) = f_{NaP}(\mathbf{z})$ for all $\mathbf{z} \in C(\mathbf{x})$. Without loss of generality, let $f_{NaP}(\mathbf{z}) = 1$. The average classification loss in $C(\mathbf{x})$ is $\ell = \Pr(y = 0 | \, \mathbf{z} \in C(\mathbf{x})) \leq 1/2$, i.e. the Bayes' optimal loss.

Now assume $f_{adv}(\mathbf{z}) = -1$ for some $\mathbf{z} \in C(\mathbf{x})$. Then for all $\mathbf{z}' \in C(\mathbf{z}) = C(\mathbf{x})$, the attacker can always transform $\mathbf{z}'$ to $\mathbf{z}$ such that $f(\mathbf{z}') = -1$. The loss under this transformation ℓ' is then $1 - \ell \geq 1/2 \geq \ell$. Recall that f_{adv} minimizes the maximum loss over all possible transformations. However, f_{NaP}'s loss over $C(\mathbf{x})$ under transformation, which is ℓ, is no more than the loss of f_{adv}, which is at least ℓ'. Then either f_{NaP} is also a minimizer to Objective 4.2, or the initial assumption in this paragraph is false. In either case, $f_{NaP}(\mathbf{z}) = f_{adv}(\mathbf{z})$. The same argument applies for $f_{NaP}(\mathbf{z}) = -1$ and thus the proof is completed. □

Remark. One example of relation satisfying the condition is a relation that only contains simple substitution. The intuition behind the theorem is that the transformation fully maintains the program semantics.

4.4.2 Required Model Capacity

The comparison in 4.4.1 assumes that the model class $\mathcal{F}$ contains all possible labeling functions. However, in practice, learning the best classifier from a large model class is typically harder than from a small one and often requires large amount of data. In this section, we show by example that *normalize-and-predict* can obtain a robust and accurate classifier on a simple model class, which is otherwise impossible using adversarial training.

This observation suggests that *normalize-and-predict* may have the advantage when we have limited data over a large feature space.

We consider a simple hypothetical setting with five APIs, which are denoted by A, B, C, D_1, D_2. API D_1 and D_2 are renamed versions of the same API D. We consider one simple relations – if a software uses any D_i for $i \in \{1, 2\}$, then it can replace D_i with any non-empty subset of $\{D_1, D_2\}$. We say a tuple of APIs is present in a software if the software uses all APIs in the tuple. A software is malicious if and only if any of the following condition is true:

1. (A, B) is present;
2. (A, D_i) is present for any $i \in \{1, 2\}$;
3. (B, C, D_i) is present for any $i \in \{1, 2\}$.

Let $\mathbf{x} \in \{0, 1\}^5$ denote the binary representation of a software by its API usage. The class label $y = +1$ for malicious instances and -1 for benign ones. Let $\mathcal{F} = \{f_{w,b} : x \to sgn(\langle w, \mathbf{x}\rangle - b) | w \in \mathbb{R}^5, b \in \mathbb{R}\}$ denote the set of all linear classifiers.

Lemma 4.4.4. *No classifier $f \in \mathcal{F}$ can classify all $\mathbf{x} \in \{0, 1\}^5$ correctly and at the same time respect the substitution relations.*

Proof. Let w_A denote the w's coordinate corresponding to API A and similarly w_B, w_C, w_{D_1}, w_{D_2} for the other four APIs. In order to classify all possible $\mathbf{x}$ correctly, the classifier $f_{w,b}$ must satisfy

$$w_A + w_C < b \tag{4.3}$$

$$\forall i \in \{1, 2\}, w_A + w_{D_i} > b \tag{4.4}$$

$$w_B + w_{D_1} + w_{D_2} < b \tag{4.5}$$

$$\forall i \in \{1, 2\}, w_B + w_C + w_{D_i} > b \tag{4.6}$$

First, by Formula 4.3 and 4.4, we have $w_{D_1} > w_C$ and $w_{D_2} > w_C$. However, by Formula 4.5 and 4.6, we have $w_{D_1} + w_{D_2} < w_C + w_{D_2}$, which implies $w_{D_1} < w_C$. Contradition. Therefore, no $f \in \mathcal{F}$ satisfies all the equations. □

On the other hand, if we perform normalization by using D to replace both D_1 and D_2, then a software is malicious if and only if

1. (A, B) is present;
2. (A, D) is present;
3. (B, C, D) is present.

Now we consider the linear model class $\mathcal{F}' = \{f_{w,b} : x \to sgn(\langle w, \mathbf{x} \rangle - b) | w \in \mathbb{R}^4, b \in \mathbb{R}\}$ over the normalized inputs.

Lemma 4.4.5. *There exists a classifier $f \in \mathcal{F}'$ that classifies all normalized input vector $\mathbf{x} \in \{0, 1\}^4$ correctly.*

Proof. We can construct such a classifier f by setting $w_A = 0.7, w_B = 0.5, w_C = 0.2, w_D = 0.4$ and finally $b = 1$. □

Through this example, we see that normalization allows to find a robust and accurate linear classifier in Lemma 4.4.5, which is not possible before normalization in Lemma 4.4.4.

4.4.3 Hardness of Training

The discussion in Section 4.4.1 and 4.4.2 is all with respect to the true optimal classifier for each approach. However, real-world machine learning is always done on a finite size data set. In Section 4.3, we have also given the formulation of the ERM problems, and the classifier obtained may be different from the true optimal.

There are two advantages of *normalize-and-predict* in the training stage. First, the guarantee of respecting relations still holds. The guarantee comes from the fact that all vectors in $\text{post}_{\mathcal{R}}(\mathbf{x})$ have the same normal form, and thus shall have the same prediction outcome for any arbitrary function f. Second, *normalize-and-predict* only requires solving a minimization problem, which in general is easier in practice than the min-max problem in adversarial training.

For adversarial training, whether robustness over training examples generalizes to robustness over test examples is still an open problem in the adversarial machine learning community, therefore the guarantee in Proposition 4.4.2 may no longer hold. Meanwhile, the performance of the adversarially trained classifier typically relies on the quality of the inner maximization subroutine in Formula 4.2. Suboptimal attack in the inner loop may cause bad performance when facing stronger attacker. We validate the performance of adversarial training in Section 6.5 over a real-world malware detection data set.

4.5 Substitution Rules and Attack Algorithms

While previous literature has developed various attacks creating adversarial examples by adding API calls, there is no attack yet in the malware detection domain using API substitutions. In this section, we introduce two attack algorithms to generate adversarial examples conforming to the simple substitution relation described in Section 4.2.3, which is an one-to-one equivalence relation. Both attacks are computationally efficient heuristics to the inner maximization problem in Formula 4.2, which is known to be NP-hard in general and is impractical to solve by exhaustive search due to high input dimensions.

4.5.1 Substitution Relations

The substitution relation class in our attacks is the one-to-one equivalence relations, which are defined as follows.

Definition 4.5.1 (One-to-one Substitution Relation)**.** *Let i, j be the indices of two APIs. A one-to-one substitution relation (i, j) exists if API j can always replace API i to realize the same malicious functionality.*

Definition 4.5.2 (One-to-one Equivalence Relation)**.** *API i and API j are said to have one-to-one equivalence relation if both substitution relation (i, j) and (j, i) exist, i.e. the two API calls are interchangeable.*

Such one-to-one substitution and equivalence relation indeed exists between many APIs, including but not limited to APIs with the same functionality in different *.dll* files and different version of the same API in a library. A detailed procedure for harnessing equivalence relations is explained in the experiment section.

All substitution relations can be stored as a list of API indices pairs shown in Definition 4.5.1 and 4.5.2. In addition, equivalence relations can also be represented and stored in the form of equivalence group defined as follows.

Definition 4.5.3 (Equivalence Group)**.** *Let $E = \{i_1, \cdots, i_k\}$ denote a set of k APIs represented by their indices. Then a group E is said to be an equivalence group if 1) i and j have one-to-one equivalence relation for all $i, j \in E$, and 2) API i and API j do not have one-to-one equivalence relation for all $i \in E$ and $j \notin E$. An API forms a stand-alone equivalence group if it is not equivalent to any other APIs.*

If many APIs are equivalent to each other, equivalence group will be a more compact representation to equivalence relation than a list of substitution tuples. In addition, the equivalence groups are a partition of the APIs in the input space.

4.5.2 Attack Algorithm

We propose two attack algorithms – GreedyByGroup and GreedyByGrad– to create adversarial examples using substitutions as shown in Algorithm 3 and 4. Both algorithms iteratively modifies the input example by applying substitution rules. The subsitutions made in each iteration are chosen to maximize some surrogate objectives, which potentially lead to higher classification loss. The difference between the two algorithms is the choice of the objective – GreedyByGroup directly uses the change in loss function, while GreedyByGrad uses the local gradients to approximate the change in loss. Since adversarial examples are only created for malware, we use $\ell_f(\mathbf{x})$ as a short hand of $\ell(\mathbf{x}, +1, f)$, the loss of a malware under classifier f.

GreedyByGroup GreedyByGroup takes the initial feature representation $\mathbf{x}$ of a malware as input and a parameter indicating maximum number of iteration. We first introduce some notations used in the algorithm. We say an equivalence group E is *present* in $\mathbf{x}$ if it contains an API that appears in $\mathbf{x}$ ($\exists i \in E$ such that $\mathbf{x}_i = 1$). Let $\mathcal{E}$ be the set of equivalence groups present, $k = |\mathcal{E}|$ be the number of equivalence groups present, and E_i be the i-th group in $\mathcal{E}$. In addition, we call $s \in \{0,1\}^{|E|}$ an assignment to a group of APIs E. Last, we use $\mathbf{x}_{E,s}$ to denote the feature vector obtained by applying an assignment s to E in $\mathbf{x}$, i.e. replacing the original feature values of APIs in E with the values in s. $s = \{0\}^{|E|}$ is not a legal assignment to apply as at least one of the API in the group must be present to ensure the functionality of the original malware is not changed. On the other hand, it is valid to apply an assignment that cause multiple APIs in a group to be present.

In each iteration, the attacker first finds all equivalence groups present in the current version of x, i.e. the set $\mathcal{E}$. For each group E_i, the attacker finds the assignment s_i such that $\mathbf{x}_{E_i,s_i}$ maximizes the loss function. At the end of each iteration, the attacker applies these assignments to all groups present to obtain a new $\mathbf{x}$. The final adversarial example is

the vector with the largest loss among the original input $\mathbf{x}$ and all $\mathbf{x}$ at the end of each iteration.

In practice, we have observed that the size of a group $|E|$ ranges from 2 to 23 and most groups contain fewer than 10 APIs. Therefore, we enumerate every possible assignment when $|E| \leq 10$ and randomly sample 1024 assignments when $|E| > 10$.

Algorithm 3 GreedyByGroup $(\mathbf{x}, MaxIter)$

$\mathbf{x}^{adv} = \mathbf{x}, k = 0$

while $k < MaxIter$ **do**

  $k = k + 1$

  Find the set $\mathcal{E}$ of all equivalence groups present in $\mathbf{x}^{adv}$

  for $E_i \in \mathcal{E}$ **do**

    $s_i = \arg\max_{s \in \{0,1\}^{|E_i|} - \{0\}^{|E_i|}} \ell_f(\mathbf{x}_{E_i,s})$

  end

  for $E_i \in \mathcal{E}$ **do** $\mathbf{x} = \mathbf{x}_{E_i,s_i}$;

  if $\ell_f(\mathbf{x}) > \ell_f(\mathbf{x}^{adv})$ **then** $\mathbf{x}^{adv} = \mathbf{x}$;

end

return $\mathbf{x}^{adv}$

Algorithm 4 GreedyByGrad$(\mathbf{x}, m, MaxIter, List_{\mathcal{R}})$

$\mathbf{x}^{adv} = \mathbf{x}, k = 0$

while $k < MaxIter$ **do**

  $k = k + 1$

  $g = \nabla_{\mathbf{x}} \ell_f(\mathbf{x})$

  for $(i, j) \in List_{\mathcal{R}}$ *where* $\mathbf{x}_i = 1$ **do**

    $c_{i,j} = \max(g_j, g_j - g_i)$

  end

  for (i, j) *corresponding to top* m *largest positive* $c_{i,j}$ **do**

    if $c_{i,j} = g_j - g_i$ **then** $\mathbf{x}_i = 0, \mathbf{x}_j = 1$;

    else $\mathbf{x}_j = 1$;

  end

  if $\ell_f(\mathbf{x}) > \ell_f(\mathbf{x}^{adv})$ **then** $\mathbf{x}^{adv} = \mathbf{x}$;

end

GreedyByGrad GreedyByGrad takes four input arguments. Vector $\mathbf{x}$ is the original test input; $List_{\mathcal{R}}$ contains all substitution relations represented in a list of tuples; m is the maximum number of subsitution rules applied in one iteration; and $MaxIter$ is the maximum number of iterations. We use g to denote the gradient of the loss function w.r.t. the input at $\mathbf{x}$, and g_i to denote the i-th coordinate of g.

In each iteration, the attacker first computes $c_{i,j}$, the potential increase in loss function of applying a substitution rule (i, j), for all applicable rules in $List_{\mathcal{R}}$. The gain is approximated by using first order Taylor expansion – the loss ℓ approximately increase by g_j if $\mathbf{x}_j$ changes from 0 to 1, and decreases by g_i if $\mathbf{x}_i$ changes from 1 to 0. We allow two outcomes of applying substution rules: either only $\mathbf{x}_j = 1$, or both $\mathbf{x}_i$ and $\mathbf{x}_j = 1$. The latter captures the case in which the attacker uses different versions of APIs at different locations to achieve the same functionality. The potential change $c_{i,j}$ is then the maximum value between g_j and $g_j - g_i$.

After computing the $c_{i,j}$, the attacker chooses m rules with the largest positive potential increase $c_{i,j}$, and apply these rules to $\mathbf{x}$. If there are fewer than m rules satisfying the condition, the attacker will apply all of these rules. The final adversarial example is the vector with the largest loss among the original input $\mathbf{x}$ and all $\mathbf{x}$ at the end of each iteration.

4.6 Experiment

We evaluate the effectiveness of our substitution attacks, GreedyByGroup and GreedyByGrad, and the effectiveness of adversarial training and *normalize-and-predict* for learning classifiers that respect a relation. Our evaluation aims to answer the following questions:

1. Do our substitution attacks pose real threats to robust malware detection?

2. Do adversarially training and *normalize-and-predict* give similar results in practice, when trained on finite training sets?

   We know from Theorem 4.4.3 that adversarial training and *normalize-and-predict* learn the same optimal classifier when given the same infinite training set conforming to a simple substitution relation.

3. *Normalize-and-predict* respects a relation by design. How do adversarially trained models respect relations?

Our evaluation results show that:

1. Our substitution attacks defeat adversarially trained models for addition attacks [ADHHO18] and pose real threat to robust malware detection. The False Negative Rate (FNR) increases from 5.6% (natural examples in the data set) to 82.8% (adversarial examples generated by our attacks).

2. Adversarial training learns classifiers that approximate the given relation. Adversarially trained models have a 12.8% FNR on the adversarial examples generated by our attacks. *Normalize-and-predict* achieves 7.4% FNR on the same adversarial examples.

3. An intuitive hypothesis for explaining how adversarially trained models respect a simple substitution relation is that the model has similar or even the same weights between inputs for equivalent APIs and the first hidden layer neurons. We performed this computation and found that it is not case. We then use prediction-attribution techniques such Integrated Gradient (IG) [STY17] and show that the adversarially trained models indeed have similar prediction attribution scores for equivalent APIs. Therefore, adversarial training approximate relations in a non-obvious way.

4.6.1 Experiment Setup

Our experiments use the same setup as the one in [ADHHO18], including the data set, the model structure, and the training and testing data split. We use their setup because their work represents state-of-the-art results against addition attack. If our substitution attacks are effective against these models that are robust against addition attacks, substitution attacks pose new threats to robust malware detection.

Data Sets The data set contrains Windows binary API usage features of 34,995 malware and 19,696 benign software, extracted from their Portable Executable (PE) files using LIEF. [4] There are 22,761 unique API calls in the data set, so each PE file is represented by a binary indicator vector $\mathbf{x} \in \{0,1\}^m$, where $m = 22,761$.

Model Structure We use a fully-connected neural net with three hidden layer, each with 300 nodes. All hidden layer nodes use the ReLU activiation function.

Data Split We sample 19,000 benign PEs and 19,000 malicious PEs to construct the training (60%), validation (20%), and test (20%) sets.

4.6.2 Procedure

Our experiment contains additional steps for performing substitution attacks and evaluating effectiveness of adversarial training and *normalize-and-predict* against substitution attacks.

1. Extract Substitution Rules: Two APIs that appear in a substitution rule have similar functionality and are interchangeable. While it is in general difficult to automatically

[4] KDD 19 adversarial malware detection competition uses data sets of the same format provided by the same authors.

extract equivalent APIs, we find it is effective to extract substituion rules based on API names. We identified four types of patterns for extracting substitution rules:

- API with the same name but located in different Dynamically Linkable Libraries (DLLs). For example, the `memcpy` is a commonly used standard C library function. In different versions of Windows and MSVC compilers, `mempcy` is shipped in libraries with differnt names, including `crtdll.dll`, `msvcr90.dll`, and `msvcr110.dll`. `crtdll.dll:memcpy`, `msvcr90.dll:memcpy`, and `msvcr110.dll:memcpy` become three different API features in the data set, but they are equivalent and interchangeable.

- API with and without the `Ex` suffix. The `Ex` suffix represents an extension to the same API without the suffix. For example, `Sleep` and `SleepEx` both realize the functionality of suspending the current running thread.

- API with and without the `A` or `W` suffixes. The `A` suffix represents the single charater version (i.e. using `char[]` for strings). The `W` suffix represents the wide character version (i.e. using `wchar_t[]` for strings).

- API with/without `_s` suffix. The `_s` suffix represents the secure version of an API.

Using these four patterns, we extracted about 15,000 rules and 2,000 substitution sets. About 500 of the sets have more than 2 APIs and the maximal set has 23 APIs.

2. Generate Substitution Adversarial Examples: For each malicious input vector, we generate one adversarial example using GreedyByGroup and one adversarial example using GreedyByGrad.

3. Perform Adversarial Training: We use GreedyByGrad to generate adversarial substitution examples for adversarial training. We cannot use GreedyByGroup for adversarial training because it take significantly longer than GreedyByGrad to generate an adversarial example.

Table 4.1: Evaluation of model accuracy and robustness to adversarial examples.

Each column represents a method for training. The "Adv. Training" column is using adversarial examples generated by GreedyByGrad. Each row represents a test set. We present the False Negative Rate (FNR) and False Positive Rate (FPR) for each combination.

	Natural		Adv. Training		Normalize and Predict		Al-Dujaili et al.	
	FNR	FPR	FNR	FPR	FNR	FPR	FNR	FPR
Natural	6.7%	9.2%	8.1%	9.1%	7.4%	7.7%	5.6%	10.1%
GreedyByGrad	92.7%	9.2%	8.1%	9.1%	7.4%	7.7%	70.6%	10.1%
GreedyByGroup	92.7%	9.2%	12.7%	9.1%	7.4%	7.7%	82.8%	10.1%

4. Evaluate models: We use natural examples and adversarial examples to evaluate natural models, models adversarially trained for addition attacks, and models adversarially trained for substitution attacks.

4.6.3 Results

Table 4.1 summarizes the results for different combinations of attack methods and training methods. We make four observations based on the results.

First, substitution attacks pose real threats to robust malware detection. Both GreedyByGroup and GreedyByGrad can successfully evade naturally trained models (FNR ¿ 90%). Adversarially trained models for addition attacks have slightly better robustness against substitution attacks compared naturally trained models, but the FNR is still high (FNR ¿ 80%). Our results call for attentions to the threat model assumed by robust malware detection research. It is commonly assumed that it is easier to add APIs than to remove APIs. Our results show that substitution is an effective way for removing APIs.

Second, the effectiveness of adversarial training depends on the quality of the ad-

versarial examples used. In our experiment, adversarial training was performed using adversarial examples generated by GreedyByGrad. When encountered with adversarial examples generated by GreedyByGroup, which is a stronger attack than GreedyByGrad, the effectivness of adversarial training decreased. This means that adversarial training is effective against attacks with similar or weaker capabilities, but not against attackers with stronger capabilities.

Third, *normalize-and-predict* achieves the same FPR for natural examples and adversarial examples generated by the two attacks. The results are not surprising given the design goal of *normalize-and-predict*. By strictly respecting the relation, any transformations based the relation makes no impact to the prediction results, regardless of the capabilities of the attackers. This is the key difference between *normalize-and-predict* and adversarial training.

Fourth, while *normalize-and-predict* guarantees robustness against a given relation, it sacrificed a little bit of accuracy. Using the natural examples for testing, the FPR of *normalize-and-predict* is higher than the FPR of the natural models. The tradeoff between accuracy and robustness is well known. In our case, the gained robustness clearly overweighs the slight drop of accuracy.

4.6.4 Interpretability Study of Adversarial Training

Our results show the importance of learning a classifier that repsects a relation. To our surprise, adversarial learning achieved good results for learning a relation in spite of its statistical nature. In this section, we perform a deep analysis of our results to better understand what exactly adversarial learning has learned.

How Many Substitution Adversarial Examples?

The more adversarial examples there are, the more likely adversarial training would have a blind spot for the attacker to succeed. For a present equivalence group E whose size is $|E|$, the number of adversarial examples can be generated from this equivalence group is $2^{|E|} - 1$. The total number of adversarial examples for an input $\mathbf{x}$ is then $\prod_i (2^{|E_i|} - 1)$, where i iterates over all present equivalence group.

Using the formula above, we calculate the number of substitution adversarial examples that can be generated for each malicious input vector in the data set. The 0, 25, 50, 75, and 100 percentiles are 10^0, 10^{23}, 10^{33}, 10^{43}, 10^{356}, respectively. Even though the attacker has a significantly smaller space for constructing adversarial examples compared to addition attacks (up to 2^m, where $m = 22,761$ in our data set), substitution attacks still give a sufficiently large space for the attacker to be successful. In addition, while our GreedyByGrad and GreedyByGroup are effectively for generating subsection adversarial examples, the size of the space indicates that there could be stronger attacks than our attacks. We have shown that the effectivness of adversarial training (with GreedyByGrad) drops when encountered with a stronger attack (GreedyByGroup). It is concerning whether adversarial training can keep up with stronger attackers.

How Does Adversarial Training Respect a Relation?

Our results show that adversarial training does quite well against the substitution attack. The natural question to ask here is "Does adversarial training actually learn a classifier that respect the relation?" We attempt to answer this question from two different angles.

Do equivalent APIs have similar first-layer weights? Intuitively, if a neural net has the same first layer weights for equivalent APIs, then the neural net will strictly respect

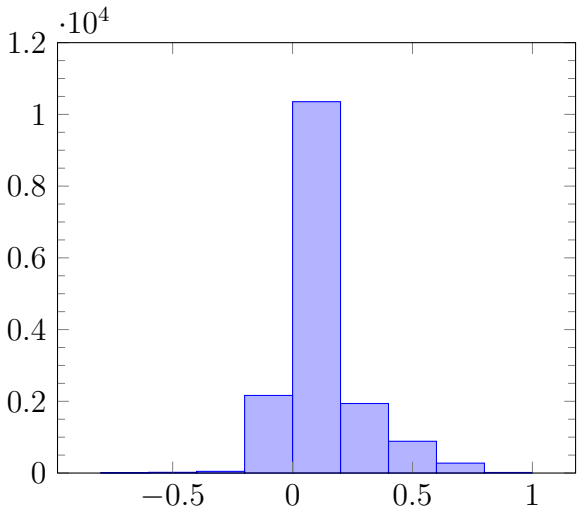

Figure. 4.1: **First layer weights difference between equivalent APIs.** Most data points are on the right side of 0.0, indicating that adversarial training does not push the weights for equivalent APIs together.

the relation. While having the same first layer weights is not a necessary condition, this is a good starting point for understanding how adversarial training respects a relation.

We calculate L_2 distance between the weight vectors for equivalent APIs. Suppose APIs i and j are equivalent. Denote w_i be the weight vector between feature i to the first hidden layer and w_j be that between feature j and the first hidden layer. We enumerate every pair of equivalent API (i, j) and calculate $||w_i - w_j||_2$ for the natural models and the adversarially trained models.

Figure 4.1 shows the histogram of $||w_i^0 - w_j^0||_2 - ||w_i^1 - w_j^1||_2$, where w^0 represents a weight vector from an adversarially trained model and w^1 represents a weight vector from a naturally trained model. The results show that for most pairs of equivalent APIs, the L_2 distance between weight vectors for the equivalent APIs and the first layer neurons increased after adversarial training. This indicates that adversarial training does not learn the relation by simply assigning equivalent or similar first layer weights. If adversarial training indeed learns the relation, it is learning the relation in a non-obvious way.

Do adversarially trained models treat equivalent APIs similarly in prediction? To answer this question, we use the Integrated Gradient (IG) [STY17], which attributes the prediction results to individual features. Suppose API i and j are equivalent. Given an input $\mathbf{x}$ where $\mathbf{x}_i = 1$ while $\mathbf{x}_j = 0$, we can use IG to calcualte an attribution score for $\mathbf{x}_i$, denoted as s_i. Then, we set $\mathbf{x}_i = 0$ and $\mathbf{x}_j = 1$, and use IG to re-calculate the attribution score for $\mathbf{x}_j$, denoted as s_j. If adversarially trained models respects the relation, s_i and s_j should be close.

Suppose $F(\mathbf{x})$ is a net that generates a prediction score in $[0, 1]$, where $\mathbf{x}$ is the input. In our case, $F(\mathbf{x})$ will be the malicious probability and $\mathbf{x}$ will be API feature vector of the input program. To better attributing prediction, the authors of IG recommend that we find a baseline input $\mathbf{x}'$, where $F(\mathbf{x}') = 0$. Typically, the baseline input is an input without any signal. For example, a black image for object identification, or zero embedding for NLP.

Interestingly, in the case of malware detection, the zero feature vector leads to high malicious score (about 0.9 for several models). Our explanation is that every software more or less needs to use some APIs; if no API is used at all, it means the programmer is intentionally avoiding using any API, which is a suspicious behavior. To find a baseline input $\mathbf{x}'$ whose $F(\mathbf{x}') = 0$, we searched in the benign software used for training, and found 11 distinctive feature vectors, whose malicious score are close to 0.

Given a baseline, the end-to-end output of IG is to assign prediction score of the input vector to individual features. The sum of the contribution scores of individual features is equal to the prediction score of the input; so $F(\mathbf{x}) = \sum_i IG(\mathbf{x}_i)$, where $IG(\mathbf{x}_i)$ represents the prediction score assigned to feature i.

For each malicious input $\mathbf{x}$, we randonly sample a pair of equivalent API (i, j),

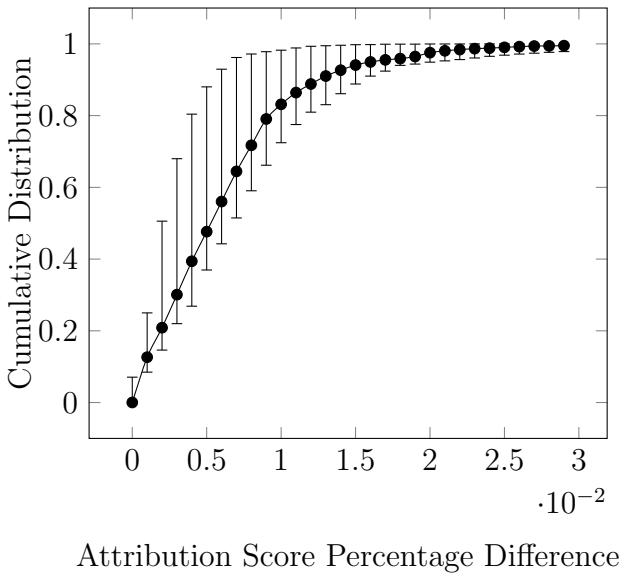

Figure. 4.2: **CDF of attribution score difference.** Since there are 11 baseline points, we present the results with an error bar for each point, presenting the maximal, median, and minimal value. The trend is consistent among different baselines used for calculating IG.

where $\mathbf{x}_i = 1$ and $\mathbf{x}_j = 0$, and then use IG to calcualte the attribution score s_i for $\mathbf{x}_i$ and s_j for $\mathbf{x}_j$.

Figure 4.2 shows the cumulative distribution function (CDF) for $|s_i - s_j|$. About 83% of the API pairs have a score difference smaller than 0.01 and about 97% of the API pairs have a score difference smaller than 0.02. Note that the malicious score should be above 0.5 to be predicted as a malware. Our results show that equivalent APIs are indeeded treated similarly by an adversarially trained models.

4.7 Conclusion

Recent research on attacks and defenses for machine learning systems has drawn concerns for using machine learning in security critical applications. Malware detection, a security application that has benefited from the prediction power of machine learning, is also fighting against adversarial attacks against machine learning systems. In this work,

we showed that the addition threat model is flawed, which assumes that it is easier to add features to malware than to remove features. We borrowed the ideas from program obfuscation and program normalization to design new attacks that remove features and evade defenses.

Our attacks and defenses are based on a theoretical framework that abstracts the impact of program transformations as relations between features. Our framework captures both the addition attacks (the monotonic relation) and our new substitution attacks (the simple substitution attack). We show that adversarial training and *normalize-and-predict* are optimizing for different objectives and prove that adversarial training and *normalize-and-predict* will learn the same optimal classifiers given infinite training data conforming a simple substitution relation.

We then empirically evaluted our substitution attacks against a state-of-the-art robust malware detection techniques. Our results showed that substitution attacks pose real threats to malware detection. We then showed *normalize-and-predict* is more effective than adversarial training against substitution attacks in practice.

A key takeaway from this chapter is that security applications should rely on techniques that can provide security guarantees. While it is tempting to use adversarial learning to automatically learn relations, our results have shown that adversarial learning only approximates the desired relation and may even violate it in the pursuit of robustness and accuracy. Such approximation leaves the door open for stronger adversaries. The *normalize-and-predict* approach on the other hand provide robustness by design regardless of the strength of the attackers.

4.8 Acknowledgement

Chapter 4 is based on the manuscript submitted to Oakland Security and Privacy 2020 (Yizhen Wang, Xiaozhu Meng, Mihai Christodorescu and Somesh Jha, "Robust Machine Learning-Based Malware Detection Against Program Transformation"). This research was conducted during an internship with Visa Research from June to September in 2018 and 2019. Mahashweta Das and Mihai Christodorescu at Visa Research contributed to this research. The dissertation author is the co-primary investigator and co-author of these materials. Somesh Jha is the main contributor to the definition of logical relation and robustness w.r.t. relation. Xiaozhu Meng is the main contributor the interpretability study of adversarial training.

Chapter 5

Data Poisoning Attack against Online Learning

5.1 Introduction

In this chapter, we initiate a systematic investigation of data poisoning attacks for online learning. We begin by formalizing the problem into two settings, semi-online and fully-online, that reflect two separate use cases for online learning algorithms. In the semi-online setting, only the classifier obtained at the end of training is used downstream, and hence the adversary's objective involves only this final classifier. In the fully-online setting, the classifiers is updated and evaluated continually, corresponding to applications where an agent continually learns and adapts to a changing environment – for example, a tool predicting price changes in the stock market. In this case, the adversary's objective involves classifiers accumulated over the entire online window.

We next consider online classification via online gradient descent, and formulate the adversary's attack strategy as an optimization problem; our formulation covers both semi-online and fully-online settings with suitable modifications and applies quite generally

to a number of attack objectives. We show that this formulation has two critical differences with the corresponding offline formulation [MZ15] that lead to a difference in solution approaches. The first is that unlike the offline case where the estimated classifier is the minimizer of an empirical risk, here the classifier is a much more complex function of the data stream which makes gradient computation time-consuming.The second difference is that data order now matters, which implies that modifying training points at certain positions in the stream may yield high benefits; this property can be potentially exploited by a successful attack to reduce the search space.

We then propose a solution that uses three key steps. First, we simplify the optimization problem by smoothing the objective function and using a novel trick called label inversion if needed. Second, we recursively compute the gradient using Chain rule. The third and final step is to narrow down the search space of gradient ascent by modifying data points at certain positions in the input stream. We propose three such modification schemes – the Incremental Attack which corresponds to an online version of the greedy scheme, the Interval Attack which corresponds to finding the best consecutive sequence to modify, and finally a Teach-and-Reinforce Attack, which seeks to teach a classifier that adheres to the adversary's goal by selectively modifying inputs in the beginning of the data stream, and then reinforce the teaching by uniformly modifying inputs along the rest of the stream.

Finally, we carry out detailed experimentation which evaluates these attacks in the context of an adversary who seeks to degrade the classification error for four data sets, two settings (semi-online and fully-online) as well as three styles of learning rates. Our experiments demonstrate that online adversaries are indeed significantly more powerful in this context than adversaries who are oblivious to the online nature of the problem. Additionally, we show that the severity of the attacks depend on the learning rate as well as the setting; online learners with rapidly decaying learning rates are more susceptible to

attacks, and so is the semi-online setting. We conclude with a brief discussion about the implication of our work for constructing effective defenses.

5.2 The Setting

5.2.1 The Classification Setting and Algorithm

We consider online learning for binary linear classification. Specifically, we have instances x drawn from the instance space $\mathbb{R}^d$, and binary labels $y \in \{-1, +1\}$. A linear classifier is represented by a weight vector w; for an instance x, it predicts a label $\text{sgn}(\theta^\top x)$. For a particular classification problem, this weight vector w is determined based on training data.

In offline learning, the weight vector θ is learnt by minimizing a convex empirical loss ℓ on training data plus a regularizer $\Omega(\theta)$. In contrast, in online learning, the training data set S arrives in a stream $\{(x_0, y_0), \ldots, (x_t, y_t), \ldots\}$; starting with an initial θ_0, at time t, the weight vector θ_t is iteratively updated to θ_{t+1} based on the current example $S_t = (x_t, y_t)$.

Since the classical work of [LLW91], there has been a large body of work in online learning in a number of settings [CBL06]. In this work, we consider online learning in a distributional setting where each (x_t, y_t) is independently drawn from an underlying data distribution $\mathcal{D}$. As our learning procedure, we select the popular Online Gradient Descent (OGD) algorithm [Zin03], which at time t, performs the following update:

$$\theta_{t+1} = \theta_t - \eta_t(\nabla \ell(\theta_t, (x_t, y_t)) + \nabla \Omega(\theta_t)),$$

Recall that here ℓ is a convex loss function, and Ω is a regularizer. For example, in the popular L_2-regularized logistic regression, ℓ is the logistic function: $\ell(\theta, (x, y)) =$

$\log(1 + e^{-y\theta^\top x})$ and $\Omega(\theta) = \frac{c}{2}\|\theta\|^2$ for some constant c. In addition, η_t is a learning rate that typically diminishes over time. We use T to denote the length of the data stream.

5.2.2 The Attacker

Following prior work on data poisoning [MZ15, KL17], we consider a white-box adversary. Specifically, the adversary knows the entire data stream (including data order), the model class, the training algorithm, any hyperparameters and any defenses that may be employed. Armed with this knowledge, the adversary has the power to arbitrarily modify at most K examples (x_t, y_t) in the stream. While this is a strong assumption, it has been used in a number of prior works [MZ15, KL17, BNL12, BL17]. Security by obscurity is well-known to be bad practice and to lead to pit-falls [BPRB$^+$13]. Additionally, adversaries can often reverse-engineer the parameters of a machine learning system [TZJ$^+$16, SSSS17].

We consider two different styles of attack objectives – *semi-online* and *fully-online* – that correspond to two different practical uses of online learning. In the sequel, we use the notation $f(\theta)$ to denote an attack objective function that depends on a specific weight vector θ.

Semi-Online. Here, the attacker seeks to modify the training data stream so as to maximize its objective $f(\theta_T)$ on the classifier θ_T obtained *at the end of training*. This setting is applicable when an online or streaming algorithm is used to train a classifier, which is then used directly in a downstream application.

Here, the training phase uses an online algorithm but the evaluation of the objective is only at the end of the stream. Compared to the classic offline data-poisoning attack setting [BNL12, SKL17], the attacker now has the extra knowledge on the order in which training data is processed.

Fully-online. Here, the attacker seeks to modify the training data so as to maximize the accumulated objectives over the entire online learning window. More specifically, the

attacker's goal is to now maximize $\sum_{t=1}^{T} f(\theta_t)$.

This setting is called fully-online because both the training process and the evaluation of the objective are online. It corresponds to adversaries in applications where an agent continually learns online, thus constantly adapting to a changing environment; an example of such a learner is a financial tool predicting price changes in the stock market.

An Example Attack Objective: Classification Error. Our algorithm and ideas apply to a number of objective functions f; however, for specificity, while explaining our attack, we will consider an adversary whose objective is the classification error achieved by the classifier output by the online learner:

$$f(\theta) = \Pr_{(x,y)\sim\mathcal{D}}(\text{sgn}(\theta^\top x) \neq y) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}[\mathbb{1}(\text{sgn}(\theta^\top x) \neq y)] \tag{5.1}$$

The Feasible Set Defense. Following prior work [BNL12, SKL17] will also assume that the learner employs a *feasible set* defense, where each example (x_t, y_t) on arrival is projected on to a feasible set $\mathcal{F}$ of bounded diameter. This defense rules out trivial attacks where outlier examples with very high norm may be used to arbitrarily modify a classifier at training time. Observe that under the feasible set defense, it is sufficient to consider adversaries which only alter training points to other points inside the feasible set $\mathcal{F}$.

For the purpose of this work, we will consider $\mathcal{F} = [-1, 1]^d \times \{-1, 1\}$.

5.3 Attack Methods

In the setting described above, finding the attacker's optimal strategy reduces to solving a certain attack optimization problem. We begin by describing this problem.

In the sequel, given a data stream S, we use the notation S_t to denote the t-th labeled example in the stream. Additionally, the difference between two data streams S

and S', denoted by $S \setminus S'$, is defined as the set $\{S_t | S_t \neq S'_t\}$; its cardinality $|S \setminus S'|$ is the number of of time steps t for which $S_t \neq S'_t$.

5.3.1 Attacker's Optimization Problem

Under this notation, the attacker's optimal strategy under an input data stream S_{train} in the semi-online case can be described as the solution to the following optimization problem.

$$\max_{S \in \mathcal{F}^T} f(\theta_T) \tag{5.2}$$

$$\text{subject to:} \quad |S \setminus S_{train}| \leq K, \tag{5.3}$$

$$\theta_t = \theta_0 - \sum_{\tau=0}^{t-1} \eta_\tau (\nabla \ell(\theta_\tau, S_\tau) + \nabla \Omega(\theta_\tau)), 1 \leq t \leq T \tag{5.4}$$

Here, (5.3) ensures that at most K examples in the input data stream are changed, and (5.4) ensures that θ_t is the OGD iterate at time t. In the fully online case, the objective changes to $\sum_{t=1}^{T} f(\theta_t)$.

5.3.2 Attack Algorithm

Challenges and Our Approach. Observe that this optimization problem has some similarities to the corresponding offline problem derived by [MZ15]. However, there are two important differences that leads to a difference in solution approaches.

The first is that unlike the offline case where the estimated weight vector is the minimizer of an empirical risk, here θ_t is a more complex function of the data stream S. This has two consequences that make gradient computation more challenging – first, changing x_t now influences all later w_τ for $\tau > t$; second, the KKT conditions that were

exploited by prior work [BNL12, SKL17, KL17] for easy computation no longer hold. The second difference from the offline case is that data order now matters, which implies that modifying training points at certain positions in the stream may yield high benefits. This property could be potentially exploited by a successful attack to reduce the search space. We next provide a solution approach that involves three steps, corresponding to three key ideas.

The first key idea is to simplify the optimization problem and make it amenable for gradient ascent; this is done by smoothing the objective function and using a novel tool called *label inversion* if needed. The second idea is to compute the gradient of the objective function with respect to points in the data stream by use of the chain rule and recursion. Finally, our third key idea is to narrow the search space by confining our search to specific positions in the data stream. We next describe each of these steps.

Idea 1: Simplify Optimization Problem. We begin by observing that in many cases, the objective function (5.2) may not be suited for gradient ascent. An example is the classification error objective function (5.1); being a 0/1 loss, it is non-differentiable, and second, it involves an expectation over $\mathcal{D}$ which we cannot compute as $\mathcal{D}$ is generally unknown. In these cases, we smooth the objective function using standard tools – we use a convex surrogate, the logistic loss, instead of the 0/1 loss, and evaluating the expectation over a separate validation data set S_{valid} [BNL12].

However, the surrogate to the classification loss objective (5.1) still has local maxima at the constant classifier. To address this, we propose a novel tool – *label inversion*. Instead of maximizing logistic loss on the validation set, we construct an inverted validation set $S_{validinv}$, which contains, for each (x, y) in the validation set, an example $(x, -y)$. We then maximize the negative of the logistic loss on this inverted validation set.

Formally, for an example (x, y) and a weight vector θ, let $L(\theta, (x, y))$ denote the

negative logistic loss $L(\theta,(x,y)) = -\log(1+\exp(-y\theta^\top x))$. For the semi-online case, the objective (5.2) reduces to $\max_{S\in\mathcal{F}^T} L(S) = \max_{S\in\mathcal{F}^T} \sum_{(x,y)\in S_{validinv}} L(\theta_T,(x,y))$. Intuitively, this encourages the attacker to modify the data stream to fit a dataset whose labels are inverted with respect to the original data distribution. Again, an analogous objective can be derived for the fully online case.

Idea 2: Compute Gradients via Chain Rule. The next challenge is how to compute the gradient of the objective function with respect to a training point (x_t, y_t) at position t. Unlike the offline case, this is more challenging as the KKT conditions can no longer be used, and the gradient depends on position t. Our second observation is that this can still be achieved by judicious use of the chain rule.

We first observe that in the computation of any θ_τ, replacing a training point (x_t, y_t) with $(-x_t, -y_t)$ leads to exactly the same result; this is due to symmetry of the feasible set as well as the OGD algorithm. We therefore choose to keep the label y_t's fixed in the training set, and only optimize over the feature vector x_ts.

Let $F(\theta_t)$ denote a (possibly) smoothed version of the objective f evaluated at the t-th iterate w_t; using Chain Rule we can write:

$$\frac{\partial F(\theta_t)}{\partial x_i} = \left\{ \begin{array}{ll} 0 & \text{if } i \geq t \\ \frac{\partial F(\theta_t)}{\partial \theta_t}\frac{\partial \theta_t}{\partial x_i} & \text{if } i = t-1 \\ \frac{\partial F(\theta_t)}{\partial \theta_t}\frac{\partial \theta_t}{\partial \theta_{t-1}} \cdots \frac{\partial \theta_{i+2}}{\partial \theta_{i+1}}\frac{\partial \theta_{i+1}}{\partial x_i} & \text{if } i < t-1. \end{array} \right\} \tag{5.5}$$

Observe that the term $\partial F(\theta_t)/\partial \theta_t$ and $\partial \theta_{i+1}/\partial x_i$ may be calculated directly, and the product $\frac{\partial F(\theta_t)}{\partial \theta_t} \cdot \frac{\partial \theta_t}{\partial \theta_{t-1}} \cdots \frac{\partial \theta_{i+2}}{\partial \theta_{i+1}}$ may be calculated via $t-i$ matrix-vector multiplications. Finally, we observe that since the gradients $\partial F(\theta_t)/\partial x_i$ and $\partial F(\theta_t)/\partial x_{i+1}$ share the prefix as shown in Appendix B.1.1 and hence the gradients $\{\partial F(\theta_t)/\partial x_i\}$ for all i may be computed in $O(Td^2)$ time. A detailed derivation of the general case, as well as specific expressions

for the surrogate loss to classification error, is presented in the Appendix B.1.2.

Idea 3: Strategic Search over Positions in the Stream. So far, we have calculated the impact of modifying a single point (x_t, y_t) on the objective function; a remaining question is which data points in the input stream to modify. Recall that in a data stream of length T, there are potentially $\binom{T}{K}$ subsequences of K inputs to modify, and a brute force search over all of them is prohibitively expensive. We present below three algorithms for strategically searching over positions in the stream for attack purposes – the Incremental Attack, the Interval Attack and the Teach-and-Reinforce Attack.

A first idea is to consider a *greedy approach*, which iteratively alters the training point in the sequence that has the highest gradient magnitude. While this has been considered by prior work for offline settings, we next derive how to do this in an online setting. This results in what we call the *Incremental Attack*.

Incremental Attack. This attack employs an iterative steepest coordinate descent approach. In each iteration, we calculate the gradient of the modified objective with respect to each single training point, and pick the (x_t, y_t) with the largest gradient magnitude. This example is then updated as:

$$x_t \leftarrow \text{proj}_{\mathcal{F}}(x_t + \epsilon \partial F / \partial x_t),$$

where ϵ is a step size parameter, F is the objective function, and proj is the projection operator. The process continues until either K distinct points are modified or convergence. The full algorithm is described in Algorithm 6 in the Appendix.

Observe however that the greedy approach has two limitations. The first is that since it does not narrow the search space using the online nature of the problem, it involves a large number of iterations and is computationally expensive. The second is that it is known to be prone to local minima as observed by prior work in the offline setting. [SKL17]

Interval Attack. To address these limitations, we next propose a novel strategy that is tailored to the online setting, called the *Interval Attack*. The idea here is to find the best *consecutive sequence* $(x_t, y_t), \ldots, (x_{t+K-1}, y_{t+K-1})$ modifying which decreases the objective function the most.

More specifically, the search process is as follows. For each $t \in \{0, \ldots, T-K-1\}$, the algorithm computes the concatenated gradient $[\partial F/\partial x_t, \cdots, \partial F/\partial x_{t+K-1}]$ and carries out the following update until convergence or until a maximum number of iterations is reached:

$$[x_t, \cdots, x_{t+K-1}] \leftarrow \text{proj}_{\mathcal{F}^K}([x_t, \cdots, x_{t+K-1}] + \epsilon[\partial F/\partial x_t, \cdots, \partial F/\partial x_{t+K-1}]),$$

where ϵ is again a step-size parameter. The value of t for which the final $[x_t, \ldots, x_{t+K-1}]$ gives the best objective (5.2) is then selected.

Teach-and-Reinforce Attack. Even though the Interval attack is faster than the Incremental, it has the highest impact on the performance of the classifiers $\theta_{t'}$ for t' close to t. In particular, it is suboptimal in the fully online case, where the objective is the sum of the losses of all θ_t. This motivates our third attack strategy – the *Teach-and-Reinforce Attack*, which seeks to first *teach* a new classifier by modifying a set of initial examples, and then *reinforce* the teaching by modifying examples along the rest of the data stream.

We split K, the number of examples to be modified, into two parts αK and $(1-\alpha)K$. We then modify the first αK points in the stream, followed by every s-th remaining point, where $s = \lceil \frac{T-\alpha K}{(1-\alpha)K} \rceil$. As in previous attacks, once the attack positions are determined, the algorithm iteratively finds the best modification through gradient ascent followed by projection to the feasible set. The full algorithm is described in Algorithm 8. Finally, the optimal value of α is determined through a grid search. Observe that $\alpha = 0$ corresponds to modifying points over an uniform grid of positions in the stream, and hence the performance

of Teach-and-Reinforce is at least as good as this case.

5.4 Experiments

We now evaluate the proposed attacks experimentally to determine their practical performance. In particular, we consider the following three questions:

1. How effective are the proposed gradient ascent-based online adversaries relative to adversaries who are oblivious to the online nature of the learning process, and adversaries that simply invert training labels?

2. Which positions in the data stream are predominantly attacked by the online adversaries?

3. How does attack performance vary with the setting (semi-online vs fully-online) and learning rate?

These questions are investigated in the context of the classification error objective for four data sets, two settings (semi-online and fully-online) as well as three styles of learning rates.

5.4.1 Experimental Methodology

Baselines. We implement the three proposed online attacks – Incremental, Interval and Teach-and-Reinforce – in both semi-online and fully-online settings. For computational efficiency, in the fully-online setting, we use the objective function $\sum_{t \in G} f(\theta_t)$ where G is a regular grid containing every 10-th integer.

Additionally, we consider two baselines – the Offline attack and the Label-flip Attack. The Offline attacker represents an adversary who is oblivious to the streaming nature of

the input. It uses the method of [BNL12] to generate K additional attack points. Since their method generates points for adding to the training data, and our input stream is fixed-length, we replace a random set of K positions in the stream with the newly generated points. The Label Flip attack selects K points from the input stream, and flips their labels. We use three strategies for selecting the positions of these points – head-flip (initial K positions in the stream), tail-flip (the final K positions), and random. To keep the figures understandable, we report the best outcome acheived by the three strategies.

Datasets. We select four datasets – a synthetic dataset consisting of a 2-dimensional mixture of spherical Gaussians, MNIST, fashion MNIST and UCI Spambase. To reduce the running time, we project both MNIST and fashion MNIST to 50 dimensions via random projections. For MNIST, we use the 1 vs. 7 classification task, and for fashion MNIST, the sandals vs. boots task.

Parameter Choice. The online learner has three parameters of interest – the initial classifier w_0, the regularization function, and the learning rate η_t. We set the initial classifier w_0 in Online Gradient Descent to a logistic regression classifier trained offline on a held-out dataset. For all cases, we use L_2 regularization $\Omega(\theta) = \frac{\lambda}{2}\|\theta\|^2$ with parameter $\lambda = 0.4$. We consider three choices for learning rate – Constant (where $\eta_t = \eta_0$), Slow Decay (where $\eta_t = \eta_0/\sqrt{t}$) and Fast Decay (where $\eta_t = \eta_0/\lambda t$) – in accordance with standard practices. η_0 is chosen to ensure that in all cases, the online learner has more than 90% test accuracy when run on the clean data stream.

The gradient based attack algorithms also involve a step size parameter ϵ, which is initially set to $\sqrt{d}/100$ where d is the data dimension, and then decays as $\sqrt{d}/[100\sqrt{1 + (t/20)}]$. Finally, the hyper-parameter α in the Teach-and-Reinforce attack is chosen by grid search from $\{0, 0.25, 0.5, 0.75\}$; observe that $\alpha = 0$ corresponds to modifying points over an uniform grid of positions in the stream.

5.4.2 Results

Figure 5.1 and 5.2 show the attack results for synthetic, MNIST and Spambase for the Slow decay learning rate; the remaining figures are in the Appendix B.2.2. Each plotted point reports average test accuracy as a function of the fraction of modified points, and is an average over 8 runs. Table 5.1 provides a qualitative description of the positions in the stream that are attacked by the different methods, with full histograms in the Appendix B.2.3.

The figures show that the Incremental and Interval attacks are overall the most effective in the semi-online case, while Teach-and-Reinforce is overall the most effective in the fully-online case. Additionally, the gradient ascent-based online attacks are highly effective for all datasets (except for the synthetic 2-d mixture of Gaussians), and have higher performance than the position-oblivious Offline attack as well as the Label Flip attack. Finally, the positions attacked by Incremental and Interval attacks do change as a function of the setting (semi-online vs. fully-online) and learning rate style. In the semi-online case, for Slow Decay and Constant, the attacks modify points towards the end of the stream, while in the fully-online case, the modified points are chosen towards the beginning. In constrast, for Fast Decay, the points modified are chosen from the beginning of the stream in both cases.

5.4.3 Discussion

We now revisit our initial questions in light of these results.

Comparison with Oblivious Adversaries and Label Flip Adversaries. Both Offline and Label Flip fail to perform as well as the best online attack. This implies that an adversary who can exploit the online nature of the problem can produce better attacks than an oblivious one. Additionally, gradient ascent procedures are indeed necessary for

high performance, and simple label flipping does not suffice.

Implications of Attack Positions. We find that the positions modified by the Incremental and Interval attacks are highly concentrated in the semi-online case. This implies that we may be able to narrow down the search space of attack positions in this case. However, such short-cuts may not yield high performance in the fully-online case.

Impact of Setting and Learning Rates. The setting (semi-online vs. fully-online) and learning rates significantly impact the results. The results there imply that first, the Incremental Attack suffers from local minima which are mitigated by Teach-and-Reinforce, and second, the semi-online setting may be easier to attack than fully-online. Additionally, the efficacy of all three online attacks is higher in the Fast Decay case than Slow Decay or Constant, which suggests that Fast Decay learning rates may be particularly vulnerable to adversarial attacks.

5.4.4 Implications for Defenses

Our experiments show that the semi-online setting is more vulnerable than the fully-online setting, and the Fast Decay learning rate is also more vulnerable than Slow Decay or Constant Decay. We conjecture that this is because the classifier w_T in the semi-online setting depends heavily on a relatively smaller number of training points than the fully-online setting. A similar explanation applies to Fast Decay; because the learning rate decays rapidly, the classifiers produced depend rather critically on the few initial points.

Based on these results, we therefore recommend the use of online methods where no one classifier depends heavily on a few input points; we conjecture that these methods would be less vulnerable to adversarial attacks. An example of such a method is the averaged stochastic gradient classifier, where the classifier used in iteration t is the average $\frac{2}{t}\sum_{s=t/2}^{t}\theta_s$. An open question for future work is to investigate the vulnerability of this

Table 5.1: Attack positions chosen by each online attack against an online learner in different settings. We use 'Start', 'End' and 'Uniform' to qualitatively represent whether most attack positions are at the beginning, end or uniformly distributed over the stream. The most frequent α is reported for Teach-and-Reinforce.

	Incremental		Interval		Teach-and-Reinforce	
	Semi-online	Fully-online	Semi-online	Fully-online	Semi-online	Fully-online
Fast Decay	Start	Start	Start	Start	0.75	0.75
Constant	End	Start/Uniform	End	Start	0	0.25/0.75
Slow Decay	End	Start/Uniform	End	Start	0	0/0.75

method.

5.5 Conclusion

We initiate the study of data poisoning attacks in the context of online learning. We formalize the problem to abstract out two settings of interest – semi-online and fully-online. In both cases, we formulate the attacker's strategy as an optimization problem and propose solution strategies. Our experiments show that our attacks perform significantly better than an attacker who is oblivious to the online nature of the input data.

There are many avenues for further investigation – such as, extending our attacks to more complex classifiers such as neural networks, and building online defenses. Finally, online learning is closely related to other problems on learning from feedback, such as contextual bandits. We believe that a very interesting open question is to expand our understanding to better understand the role and capabilities of adversaries in these systems.

5.6 Acknowledgement

Chapter 5 is based on the material in Manuscript (Yizhen Wang and Kamalika Chaudhuri, "Data Poisoning Attack against Online Learning"). The dissertation author is

the primary investigator and co-author of this material.

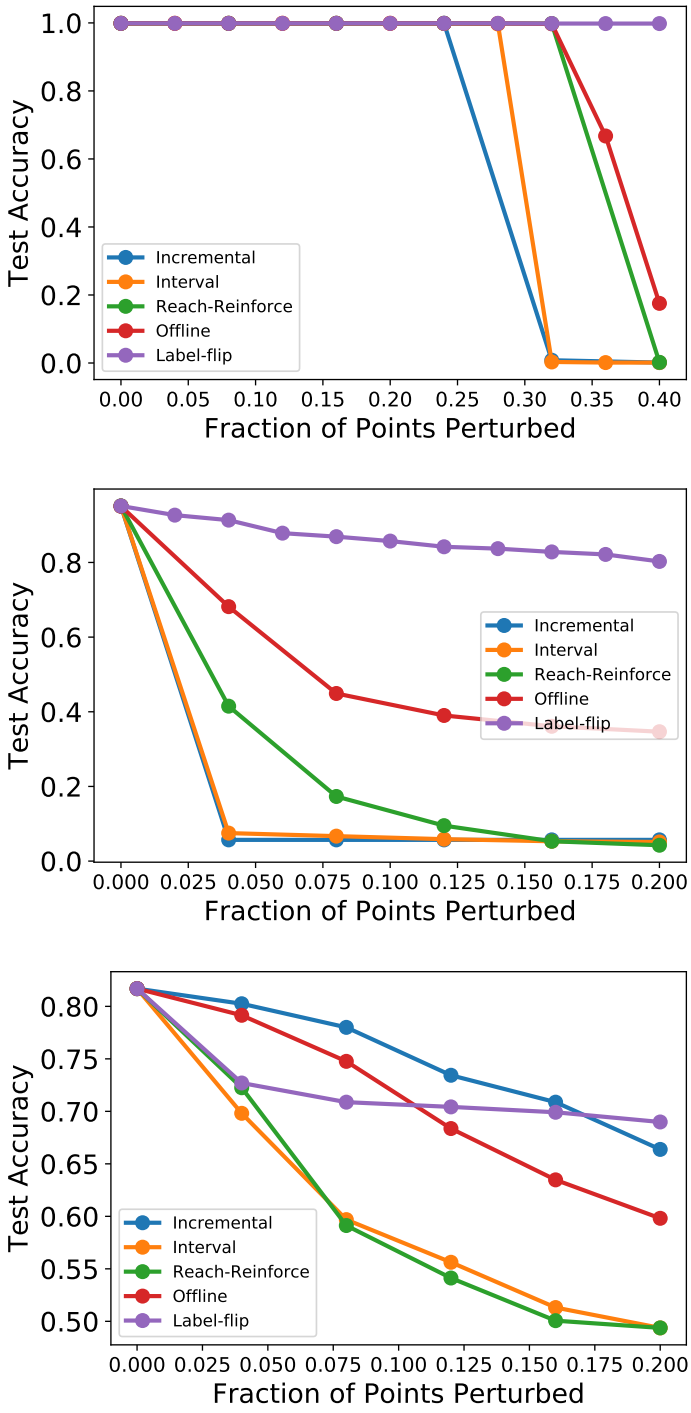

Figure. 5.1: Test accuracy vs. fraction of modified points for each attack on an online learner with Slow Decay learning rate. Semi-Online. **Top to Bottom**: 2D Gaussian mixture, MNIST 1 v.s. 7, UCI Spambase.

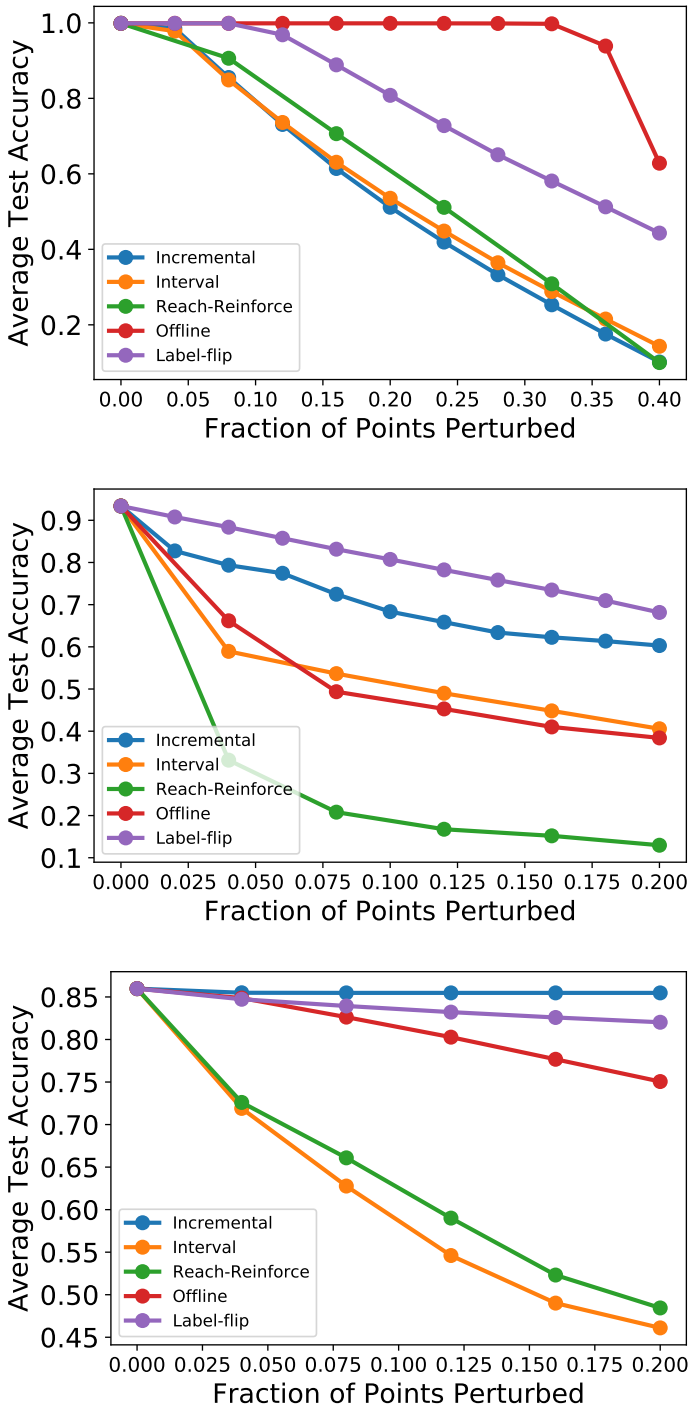

Figure. 5.2: Test accuracy vs. fraction of modified points for each attack on an online learner with Slow Decay learning rate, Fully-Online. **Top to Bottow**: 2D Gaussian mixture, MNIST 1 v.s. 7, UCI Spambase.

Chapter 6

Data Poisoning Defenses for Online Learning

6.1 Introduction

In this chapter, we consider data poisoning for online learning, and carry out a theoretical and empirical study of its effectiveness under four popular defenses. In online learning, examples arrive sequentially, and at iteration t, the learner updates its parameters based on the newly-arrived example. We select as our learner perhaps the most basic yet widely-used algorithm – online gradient descent (OGD). Starting with an initial parameter value and a loss function, OGD updates its parameter value by taking a small step against the gradient of the loss at the current example.

We consider two threat models. First, for its amenability to analysis, we consider a powerful semi-online threat model by [WC18], where the adversary can add examples in any position in the stream, and their goal is to attack the classifier produced at the end of learning. In addition, we consider a messier but more realisitic fully-online threat model – where the adversary can only add examples at pre-specified positions and their goal is to

achieve high overall loss throughout the entire stream. We assume that the learner has a defense in place that filters out input examples with certain properties determined by the defense mechanism.

For the semi-online threat model, we assume that the adversary has a target classifier in mind, and we measure how rapidly it can cause the learnt classifier to move towards the target when a specific defense is in place. We look at three regimes of effectiveness. In the first easy regime, there is a simplistic attack that the adversary can use to rapidly reach its target. In the second hard regime, there are no successful data poisoning attacks that succeed against a defense. In between lies an intermediate regime where effective attacks exist, but are not as simplistic or powerful. Specifically, our contributions are:

- We prove that the L_2-norm defense, which filters out all examples outside an L_2-ball is always in the easy regime and hence mostly weak.
- We provide conditions under which the labeling oracle defense [SHN+18, SMK+18], where the adversary *only* provides an unlabeled example to be labeled by an annotator, is in the easy regime or in the hard regime.
- We characterize the performance of two data-dependent defenses – the L_2-distance-to-centroid defense and the Slab defense [SKL17, KSL18] – when initialized on clean data.
- We empirically investigate the extent to which our theoretical observations translate to practice, and show that existing attacks are mostly effective in the regime that our theory predicts as easy, and mostly ineffective in those predicted as hard.

While our theory directly measures the speed of poisoning, a remaining question is how the poisoning attacks and defenses interact with the learning process overall – as defenses also filter out clean examples. To understand this, we consider the interplay of

poisoning and learning in a more realistic threat model – the fully online model – where the adversary can only add examples in pre-specified positions, and its goal is to force high overall loss over the entire stream. Our findings here are as follows:

- Theoretically, we show two examples that illustrate that the adversary's success depends on the "easiness" of the learning problem – for low dimensional data with well-separated classes, defenses work well, while the adversary can succeed more easily for higher dimensional, lower margin data.
- Experimentally, we corroborate that data poisoning defenses are highly effective when the classification problem is easy, but the findings from the semi-online case carry over when the classification problem is more challenging.

Our results have two major implications for data poisoning in the online setting. First, they indicate that the Slab defense may be a highly effective defense overall. Second, our experiments indicate that for challenging classification problems, weaker threat models can still result in fairly powerful attacks, thus implying that data poisoning is a threat even for these weaker adversaries.

6.2 The Setting

There are two parties in a data poisoning process – the learner and the attacker. We describe the learning algorithm as well as the attacker's capability and goals of two threat models – semi-online and fully-online.

6.2.1 Learning Algorithm

We consider online learning – examples (x_t, y_t) arrive sequentially, and the learner uses (x_t, y_t) to update its current model θ_t, and releases the final model θ_T at the con-

clusion of learning. Our learner of choice is the popular online gradient descent (OGD) algorithm [SS12, Haz16]. The OGD algorithm, parameterized by a learning rate η, takes as input an initial model θ_0, and a sequence of examples $S = \{(x_0, y_0), \ldots, (x_{T-1}, y_{T-1})\}$; at iteration t, it performs the update:

$$\theta_{t+1} = \theta_t - \eta \nabla \ell(\theta_t, x_t, y_t)$$

where ℓ is an underlying loss function – such as square loss or logistic loss. In this work, we focus on logistic regression classifiers, which uses logistic loss.

6.2.2 Threat Models

We consider two types of attackers – **semi-online** and **fully-online** – which differ in their poisoning power and objectives.

Semi-online. A semi-online attacker adds poisoning examples to the clean data stream S, and the resulting poisoned stream S' becomes the input to the online learner. We consider a strong attacker that knows the entire clean data stream and can add up to K examples to S at *any* position. Let θ be the final model output by the learner. A semi-online attacker's goal is to obtain a θ that satisfies certain objectives, for example θ could be equal to a specific θ^*, or have high error on the test data distribution. The attacker is 'semi-online' because the learner is online but 1) the attacker's knowledge of the stream resembles an offline learner's, and 2) the objective only involves the final model θ.

Fully-online. Unlike a semi-online attacker which knows the entire data stream and can poison at any position, a fully-online attacker only knows the data stream up to the current time step and can only add poisoning examples at pre-specified positions. This is a more realistic setting in which the learner gets data from poisoned sources at specific time steps. Let S' be the resulting poisoned stream of length T' and $I = \{t_1, \cdots, t_K\}$ be

the set of time steps with poisoned examples. The fully-online attacker aims to increase the online model's loss over *clean* examples in the data stream over the entire time horizon, i.e. to maximize $\sum_{t=0}^{T'-1} \ell(\theta_t, x_t, y_t)\mathbb{1}[t \notin I]$.

Note that the attacks are white-box in both settings, which is a common assumption in data poisoning. The only work in a general black-box setting is by [DHPZ19], whose methods are computationally inefficient. Other work either considers simple and limited attacker model [ZAGZ17] or knows the model class and learning algorithm [LDL+17]. The white-box setting also rules out security by obfuscation.

Defense. In both semi-online and fully-online settings, we assume that the attacker works against a learner equipped with a defense mechanism characterized by a feasible set $\mathcal{F}$. For example, $\mathcal{F}$ is an L_2-ball of radius R in the popular L_2-norm defense. If the incoming example $(x_t, y_t) \in \mathcal{F}$, then it is used for updating θ_t; otherwise it is filtered out. The attacker knows $\mathcal{F}$ under the white-box assumption.

6.3 Analysis

In this section, we analyze the effectiveness of common defenses against data poisoning in the semi-online setting. We formalize a semi-online attacker as follows.

Definition 6.3.1 (Data Poisoner)**.** *A Data Poisoner $DP^{\eta}(\theta_0, S, K, \mathcal{F}, \theta^*, \epsilon)$, parameterized by a learning rate η, takes as input an initializer θ_0, a sequence of examples S, an integer K, a feasible set $\mathcal{F} \subseteq \mathcal{X} \times \mathcal{Y}$, a target model θ^*, and a tolerance parameter ϵ and outputs a sequence of examples $\tilde{S}$. The attack is said to succeed if $\tilde{S}$ has three properties – first, an OGD that uses the learning rate η, initializer θ_0 and input stream $\tilde{S}$ will output a model θ s.t. $\|\theta^* - \theta\| \leq \epsilon$; second, $\tilde{S}$ is obtained by inserting at most K examples to S, and third, the inserted examples lie in the feasible set $\mathcal{F}$.*

For simplicity of presentation, we say that a data poisoner outputs a model θ if the

OGD algorithm obtains a model θ over the data stream $\tilde{S}$.

In order to test the strength of a defense, we propose a simple data poisoning algorithm – the SimplisticAttack. A defense is weak if this attack succeeds for small K, and hence is able to achieve the poisoning target rapidly. In contrast, a defense is strong if no attack with any K can achieve the objective. We analyze the conditions under which these two regimes hold for an online learner using logistic loss for binary classification tasks.

Algorithm 5 SimplisticAttack(θ_0, S, K, $\mathcal{F}$, θ^*, ϵ, R)

Input: initial model θ_0, clean data stream S, max number of poisoning points K, feasible set $\mathcal{F}$, target model θ^*, tolerance parameter ϵ, max L_2 norm R of inputs.
Initialize $\widetilde{\theta}_0$ as the model learned over S.
$\lambda = \|\theta^*\|, \tilde{S} = S, t = 0$
$\gamma_0 = \min\left(R/\|\widetilde{\theta}_0 - \theta^*\|, 1/\eta\right)$
while $t < K$ and $\|\widetilde{\theta}_t - \theta^*\| \geq \epsilon$ **do**
  $\gamma_t = \min(\gamma_t^*, \gamma_0)$, where γ_t^* is the solution of γ to $\frac{\gamma}{1+\exp(\widetilde{\theta}_t^\top(\theta^* - \widetilde{\theta}_t)\gamma)} = \frac{1}{\eta}$
  $(x_t, y_t) = (\gamma_t(\theta^* - \widetilde{\theta}_t), +1)$
  Find the closest $c \in [0, R/\|x_t\|]$ to 1, s.t. $(cx_t, y_t) \in \mathcal{F}$ or $(-cx_t, -y_t) \in \mathcal{F}$.
  if $(cx_t, y_t) \in \mathcal{F}$ **then** append (cx_t, y_t) to $\tilde{S}$
  else append $(-cx_t, -y_t)$ to $\tilde{S}$
  $t = t + 1$
endwhile
return $\tilde{S}$

The SimplisticAttack algorithm is described in Algorithm 5. Suppose the online learner has an initial model θ_0 and obtains a model $\widetilde{\theta}_0$ after learning over the clean stream S. SimplisticAttack iteratively appends poisoning points to the clean stream S, where all poisoning points are in the direction of $\theta^* - \widetilde{\theta}_0$ with adaptive L_2 magnitude capped at R. The algorithm is inspired by the iterative teaching method [LDH+17] and is modified with an additional projection step for general feasible set $\mathcal{F}$. In order to maintain the direction of the poisoning points, the projection step scales the input by a nonnegative factor c.

6.3.1 Bounded L_2 Norm Defense.

In many applications, the learner requires the input vector to be within a bounded domain, for example having bounded L_p norm. We consider inputs with bounded L_2 norm, which corresponds to a defense mechanism with a feasible set $\mathcal{F} = \{(x, y) | \|x\|_2 \leq R\}$ for some constant R. Notice that if the input domain is bounded by L_∞ or L_1 norms, we can still find an L_2 ball inscribed in the domain. We next show that SimplisticAttack is a very efficient data poisoner for this setting.

Theorem 6.3.2. *Let θ_0 and θ^* be the initial and the target model. Suppose the feasible set $\mathcal{F} = \{(x, y) | \|x\|_2 \leq R\}$ and $\|\theta^*\| = \lambda$. If $K > C \log(\lambda/\epsilon)$ for some constant $C = C(R)$, then* SimplisticAttack$(\theta_0, S, K, \mathcal{F}, \theta^*, \epsilon, R)$ *outputs a θ such that $\|\theta - \theta^*\| \leq \epsilon$.*

Remark. Notice that K is an upperbound of number of poisoning examples needed to output a θ close to θ^*, therefore also corresponds to a lowerbound on the speed of poisoning. Theorem 6.3.2 suggests that SimplisticAttack outputs a θ close to θ^* within logarithmic number of steps w.r.t. $1/\epsilon$. Therefore, the defense is in the easy regime. The factor C increases with decreasing R, and is in the order of $O(1/\eta R)$ when R is small. However, in real applications, the learner cannot set R to be arbitrarily small because this also rejects clean points in the stream and slows down learning. We investigate this in more details in our evaluation in Sec 6.5.

6.3.2 The Labeling Oracle Defense

In many applications, the adversary can add unlabeled examples to a dataset, which are subsequently labeled by a human or an automated method. We call the induced labeling function a labeling oracle. Denoting the labeling oracle by g, we observe that this oracle,

along with bounded L_2 norm constraint on the examples, induces the following feasible set:

$$\mathcal{F} = \{(x, y) | \|x\| \leq R, y = g(x)\}, \tag{6.1}$$

For simplicity, we analyse the oracle defense on a different feasible set $\mathcal{F}' = \{(yx, +1) | (x, y) \in \mathcal{F}\}$ derived from $\mathcal{F}$. We call $\mathcal{F}'$ the *one-sided* form of $\mathcal{F}$ because it flips all $(x, -1) \in \mathcal{F}$ into $(-x, +1)$ and as a result only contains points with label $+1$. Lemma C.1.1 in the appendix shows that an attacker that outputs θ using K points in $\mathcal{F}$ always corresponds to some attacker that outputs θ using K points in $\mathcal{F}'$ and vice versa. Therefore, the defenses characterized by $\mathcal{F}$ and $\mathcal{F}'$ have the same behavior.

We next show that for feasible sets $\mathcal{F}'$ of this nature, three things can happen as illustrated in Figure 6.1. First, if $\mathcal{F}'$ contains a line segment connecting the origin O and some point in the direction of $\theta^* - \widetilde{\theta}_0$, then SimplisticAttack can output a θ close to θ^* rapidly. Second, if $\mathcal{F}'$ is within a convex region $\mathcal{G}$ that does not contain any point in the direction of $\theta^* - \theta_0$, then no poisoner can output θ^*. Third, if $\mathcal{F}'$ contains points in the direction of $\theta^* - \widetilde{\theta}_0$ but not the origin, then the attack can vary from impossible to rapid. Theorem 6.3.3 captures the first and the second scenarios, and Appendix C.1.4 shows various cases under the third.

Theorem 6.3.3. *Let* $L(r, \mathbf{u}) = \left\{ c\mathbf{u}/\|\mathbf{u}\| \,\middle|\, 0 < c \leq r \right\}$ *denote a line segment connecting the origin and* $r\mathbf{u}/\|\mathbf{u}\|$ *for some vector* $\mathbf{u}$*. Also, let* θ_0, θ^* *be as defined in Theorem 6.3.2 and* $\widetilde{\theta}_0$ *be the online learner's model over the clean stream* S*. Suppose* $\|\theta^*\| = \lambda$*.*

1. *If* $L(r, \theta^* - \widetilde{\theta}_0) \subseteq \mathcal{F}'$ *for some* $r > 0$ *and* $K \geq C \log(\lambda/\epsilon)$ *for some constant* C*, then* SimplisticAttack$(\theta_0, S, K, \mathcal{F}', \theta^*, \epsilon, r)$ *outputs a* θ *with* $\|\theta - \theta^*\| \leq \epsilon$*.*

2. *If there exists a convex set* $\mathcal{G}$ *such that 1)* $\mathcal{F}' \subseteq \mathcal{G}$*, and 2)* $\mathcal{G} \bigcap L(+\infty, \theta^* - \theta_0) = \emptyset$*, then no data poisoner can output* θ^* *for any* K*.*

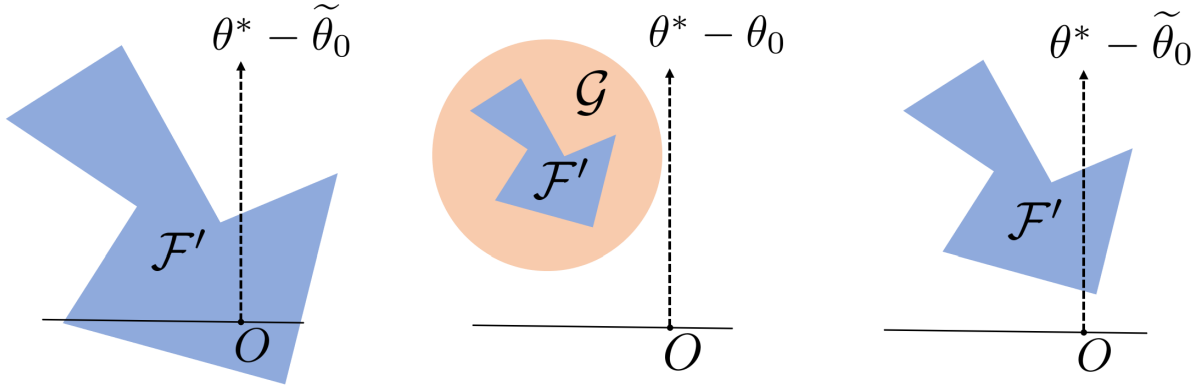

Figure. 6.1: Schematic illustration of the one-sided feasible sets $\mathcal{F}'$ corresponding to three defense regimes. Left to right: defense is in the easy, hard and intermediate regime.

6.3.3 Data-driven Defenses

A common data poisoning defense strategy is to find a subset of the training points that are anomalous or "outliers", and then sanitize the data set by filtering them out before training. Following [CSL+08], [SKL17], [PMGGL18] and [KSL18], we assume a defender who builds the filtering rule using a clean initialization data set that the poisoner cannot corrupt. Thus the poisoner knows the defense mechanism and parameters, but has no way of altering the defense. We focus on two defenses, the L_2-distance-to-centroid defense and the Slab defense [SKL17, KSL18], which differ in their outlier detection rules.

Defense Description. Both defenses assign a score to an input example, and discard it if its score is above some threshold τ. The L_2 norm constraint on the input vector still applies, i.e. $\|x\| \leq R$. Let μ_+ denote the centroid of the positive class and μ_- the centroid of the negative class computed from the initialization set.

L_2-distance-to-centroid defense assigns to a point (x, y) a score $\|x - \mu_y\|$. This induces a feasible set $\mathcal{F}_+^c = \{(x, +1) | \, \|x - \mu_+\| \leq \tau, \|x\| \leq R\}$ for points with label $+1$ and $\mathcal{F}_-^c = \{(x, -1) | \, \|x - \mu_-\| \leq \tau, \|x\| \leq R\}$ for points with label -1. The entire feasible set $\mathcal{F}^c = \mathcal{F}_+^c \bigcup \mathcal{F}_-^c$.

Slab defense assigns to a point (x, y) a score $|(\mu_+ - \mu_-)^\top (x - \mu_y)|$. We call $\beta = \mu_+ - \mu_-$ the "defense direction". Intuitively, the defense limits the distance between the input and the class centroid in the defense direction. The defense induces a feasible set $\mathcal{F}^s_+ = \{(x, +1) || \beta^\top (x - \mu_+)| \leq \tau, \|x\| \leq R\}$ for points with label $+1$ and $\mathcal{F}^s_- = \{(x, -1) || \beta^\top (x - \mu_-)| \leq \tau, \|x\| \leq R\}$ for points with label -1. The entire feasible set $\mathcal{F}^s = \mathcal{F}^s_+ \bigcup \mathcal{F}^s_-$.

L_2-distance-to-centroid Defense. If τ is large such that either $\mathcal{F}^c_+$ or $\mathcal{F}^c_-$ contains the origin, then SimplisticAttack can succeed rapidly. If τ is small such that the one-sided form of $\mathcal{F}^c$ does not contain any point on the positive side of the hyperplane with normal vector $(\theta^* - \theta_0)$, then no attack is possible. Lemma 6.3.4 shows the condition for these two cases.

Lemma 6.3.4. *Let u_+ be μ_+'s projection on $\theta^* - \theta_0$. Similarly, let u_- be μ_-'s projection on $\theta_0 - \theta^*$.*

1. *If $\tau > \min(\|\mu_+\|, \|\mu_-\|)$ and $K \geq C \log(\lambda/\epsilon)$ for some constant C, then $\exists r > 0$ such that* SimplisticAttack$(\theta_0, S, K, \mathcal{F}^c, \theta^*, \epsilon, r)$ *outputs a θ with $\|\theta - \theta^*\| \leq \epsilon$.*

2. *Otherwise, if $\langle \mu_+, \theta^* - \theta_0 \rangle < 0$, $\langle \mu_-, \theta_0 - \theta^* \rangle < 0$ and $\tau \leq \min(\|u_+\|, \|u_-\|)$, then no data poisoner can output θ^* for any K.*

Slab Defense. Each of $\mathcal{F}^s_+$ and $\mathcal{F}^s_-$ is a 'disc' between two hyperplanes with normal vector β. When τ is large such that either $\mathcal{F}^s_+$ or $\mathcal{F}^s_-$ contains the origin, SimplisticAttack can succeed rapidly. When τ is small and the projection of $y(\theta^* - \theta_0)$ on β is opposite to μ_y, then no attack is possible. Lemma 6.3.5 shows the condition for both cases.

Lemma 6.3.5. *Let $\beta = (\mu_+ - \mu_-), b_+ = -\beta^\top \mu_+, b_- = \beta^\top \mu_-$. WLOG, assume $\beta^\top (\theta^* - \widetilde{\theta_0}) \geq 0$.*[1]

[1] If $\beta^\top (\theta^* - \widetilde{\theta_0}) < 0$, we can set $\beta = (\mu_- - \mu_+)$. The Slab score remains the same.

1. *If* $\tau - b_+ > 0 > -\tau - b_+$ *or* $\tau - b_- > 0 > -\tau - b_-$*, then there exist a data poisoner* $DP^\eta(\theta_0, S, K, \mathcal{F}^s, \theta^*, \epsilon)$ *that outputs a* θ *with* $\|\theta - \theta^*\| \leq \epsilon$ *for* $K \geq C \log(\lambda/\epsilon)$ *for some constant* C*.*

2. *If* $0 \geq \tau - b_+ > -\tau - b_+$ *and* $0 \geq \tau - b_- > -\tau - b_-$*, then* SimplisticAttack *cannot output* θ^* *for any* K*. In addition, if* $\beta^\top(\theta^* - \theta_0) \geq 0$*, then no data poisoner outputs* θ^* *for any* K*.*

6.4 The Fully-Online Setting

Our theory in Section 6.3 discusses the effectiveness of attack and defense in the semi-online setting. Do effective semi-online attacks always lead to effective fully-online attacks, where the attacker wants the online model to have high loss over the entire time horizon? We show by two examples that the result varies on different learning tasks, and thus the existence of effective semi-online attack alone is not sufficient. The poisoning effect is quickly negated by clean points in one case but not in the other, which suggests the effectiveness of fully-online attack also critically depends on the difficulty of the learning tasks. As a result, the learner also needs to consider its defense mechanism's impact on the learning process.

Effectiveness of Poisoning on Different Tasks. We consider the following learner-attacker pair. The learner uses the standard OGD algorithm with initial value $\theta_0 = 0$, learning rate $\eta = 1$, and also an L_2 norm defense with $\mathcal{F} = \{(x, y) | \, \|x\| \leq 1\}$, which is always in easy regime for semi-online attacks as shown in Theorem 6.3.2. The attacker can inject 10% as many poisoning points as the clean points at any position. We then show two different tasks in Lemma 6.4.1 and 6.4.2, in which fully-online attack is hard and easy, respectively.

Lemma 6.4.1 (hard case)**.** *Consider a data distribution with input* $x \in \{1, -1\}$ *and label* $y = sgn(x)$*. The attacker cannot cause classification errors on more than* 10% *of the clean examples.*[2]

Lemma 6.4.2 (easy case)**.** *Consider an input space* $\mathbb{R}^d$ *with* $d = 10,000$*. The clean inputs are uniformly distributed over* $2d$ *points* $\{e_1, \cdots, e_d, -e_1, \cdots, -e_d\}$*, where* e_i *is the* i*-th basis vector. The label* $y = +1$ *if* $x \in \{e_1, \cdots, e_d\}$*, and* $y = -1$ *otherwise. Then the attacker can, with probability more than* 0.99*, cause classification error on* $\geq 50\%$ *of the first* $1,000,000$ *clean examples.*

In the first case, learning is easy as each clean example contains all the information needed, while in the second, learning is hard as each clean example only contributes to one dimension among many. A fully-online attack is effective if poisoning is easier than learning, and is less likely to succeed otherwise. We further validate this finding over different real-world data sets with different complexity for learning in Section 6.5.

Impact of Defenses. In reality, the defense mechanism filters both the clean and the poisoning points. Therefore, the learner cannot set the feasible set arbitrarily small since such defense also prohibits learning useful patterns. For example, if the L_2 norm bound is less than 1 for the learner in Lemma 6.4.1 and 6.4.2, then all clean points will be filtered out, and the model will be solely determined by poisoning points. The learner needs to find an appropriate defense parameter so the feasible set keeps enough clean points while slows the attack. We evaluate the effectiveness of defenses over a range of defense parameters in Section 6.5.

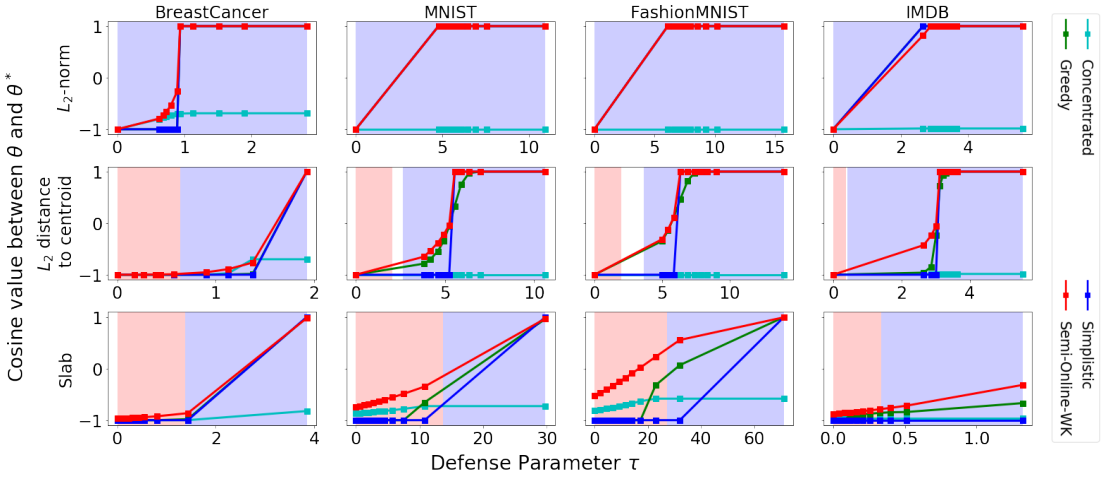

Figure. 6.2: The plot of $\cos(\theta, \theta^*)$ against defense parameter τ for semi-online attacks with a fixed number of poisoning points. Red background region indicates hard regime; blue background indicates easy regime. **Top to bottom**: L_2-norm, L_2-distance-to-centroid and Slab. **Left to Right**: BreastCancer, MNIST, FashionMNIST, IMDB.

6.5 Experiments

Is our analysis on the rapidity of attacks confirmed by practice? And how does the online classification error rate vary across different defenses and real-world data sets, given that the relative speed of poisoning and learning matters? We now investigate these questions through an experimental study. In particular, we ask the following questions.

- Do practical data-poisoning defenses exhibit the different regimes of effectiveness that we see in theory in the semi-online setting?
- How successful are fully-online attacks across different data sets, given that the tasks have different difficulty levels of learning?

These questions are considered in the context of three defenses – L_2-norm, L_2-distance-to-centroid and Slab.[3] The defenses do display different regimes of effectiveness as

[2]The bound is tight as an attacker can always cause as many errors as the number of poisoning points under L_2 norm defense.

[3]The labeling oracle defense is not included as we lack the appropriate labeling functions for the real-world tasks involved.

predicted, and the loss caused by fully-online attacks is more significant on data sets that are harder to learn.

6.5.1 Semi-online Experiment Methodology

Baseline Attacks. To evaluate the effectiveness of the defenses, we consider four canonical baseline attacks. All attacks we use append the poisoning examples to the end of the clean data stream, as [WC18] shows that this works best for our setting.

The **Simplistic** attack uses the SimplisticAttack algorithm. The **Greedy** attack [LDH+17], finds, at step t, the example $(x_t, y_t) \in \mathcal{F}$ that minimizes $\|\theta_{t+1} - \theta^*\|$, where θ_{t+1} is derived from the OGD update rule. The **Semi-Online-WK** attack [WC18] finds K poisoning examples together by maximizing the loss of the resulting model on a clean validation dataset. The optimization problem is solved using gradient descent. The poisoning points are again inserted at the end. As a sanity-check, we also include an offline baseline – the **Concentrated** attack [KSL18], which is effective against many offline defenses. We explain its adaption to the online setting in Appendix C.2.2.

Data Sets. We consider four real-world data sets – UCI Breast Cancer (dimension $d = 9$), IMDB Reviews [MDP+11] ($d = 100$),[4] MNIST 1v7 ($d = 784$) and FashionMNIST [XRV17] Bag v.s. Sandal ($d = 784$). The standard OGD algorithm is able to learn models with high accuracy on all clean data sets. Each data set is split into three parts: initialization (used for the data-driven defenses), training and test sets. Procedural details are in Appendix C.2.1.

Experimental Procedure. The learner starts with $\theta_0 = 0$ and uses the OGD algorithm to obtain a model θ by iterating over the sequence of examples output by the poisoner. The attacker sets the target model $\theta^* = -\tilde{\theta}_0$, where $\tilde{\theta}_0$ is the learner's final model if it updates only on the clean stream. Poisoning examples are restricted inside the

[4]The features are extracted using Doc2Vec on the unlabeled reviews by setting $d = 100$.

feasible set induced by the learner's defense, and scores for the data-driven defenses are calculated based on the initialization set. The number of poisoning examples K is 80 for Breast Cancer, 100 for MNIST/FashionMNIST and 200 for IMDB.

To avoid confusion by scaling factors, we evaluate the effect of poisoning by $\cos(\theta, \theta^*)$, the cosine similarity between the final model θ and the target model θ^*. Successful attacks will result in large $\cos(\theta, \theta^*)$ values.

For L_2-norm defense, the defense parameter τ is the maximum L_2 norm of the input; for L_2-distance-to-centroid, τ is the maximum L_2 distance between an input to its class centroid, and for Slab, it is the maximum Slab score. Smaller τ means stronger defense. We consider ten τ values. The i-th value is the $10i$-th percentile line of the corresponding measure for points in the clean stream. We choose τ data-dependently to ensure the range of τ is practical.

6.5.2 Fully-online Experiment Methodology

Baseline attacks and data sets. Existing fully-online attack methods [WC18, ZZ19] are either too computationally expensive for high dimensional data and long time horizons, or require knowledge of the entire clean stream. Instead, we use a generic fully-online attack scheme which reduces to semi-online attacks as follows. At time t when the learner gets examples from the attacker, the attacker runs a semi-online attack algorithm with θ_t as the initial value and a model with high fully-online loss as the target θ^* to generate the poisoning examples.[5] We construct three fully-online attackers using **Simplistic**, **Greedy** and **Semi-Online-WK** as the semi-online subroutines respectively, and name the baselines after their semi-online subroutines. We use the same four data sets as in the semi-online experiment.

[5]Such attack does *not* require knowledge of future clean points, which either needs to be given or estimated in previous methods.

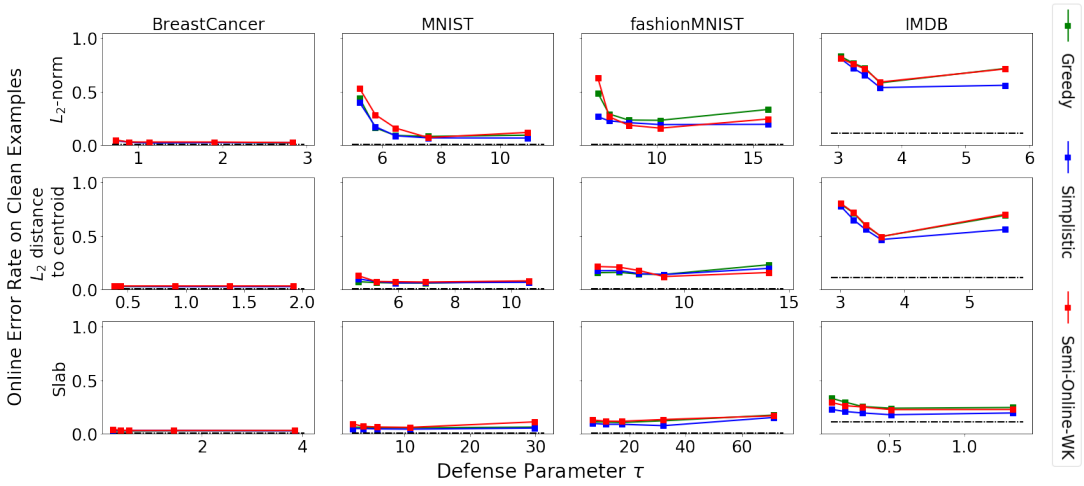

Figure. 6.3: Online classification error rate against defense parameter τ when 10% of the examples in the data stream comes from the attackers. The dashdot line shows the error rate of the offline optimal classifier. Small τ means the defense filters out more examples, both clean and poisoned. Larger online classification error means more successful attack. **Top to bottom**: L_2-norm, L_2-distance-to-centroid and Slab defense. **Left to Right**: BreastCancer, MNIST, FashionMNIST, IMDB.

Experiment Procedure. The sequence length T is 400 for Breast-Cancer and 1000 for MNIST, FashionMNIST and IMDB. For each run of experiment, a set of attack position I with $|I| = 0.1T$ is first drawn uniformly random over all possible positions.[6] The attacker generates a poisoning example when $t \in I$, otherwise a clean sample is drawn from the training set. The learner starts with $\theta_0 = 0$ and updates on the poisoned data stream. Examples that fall outside the feasible set are not used for updating model. At each iteration t, the learner also predicts the class label of the input x_t as $f_t(x_t) = \text{sgn}(\theta_t^\top x_t)$. The effect of poisoning is measured by the online classification error rate over the entire time horizon $\sum_{i=0}^{T-1} \mathbb{1}(f_t(x_t) \neq y_t)\mathbb{1}(t \notin I)$.

For each defense, we use five τ values, corresponding to five feasible sets $\mathcal{F}$ which contains 30%, 50%, 70%, 90% and 100% of the clean examples in the stream.

[6]We also try a more powerful poisoner with $|I| = 0.2T$; the result is shown in Appendix C.2.3 and has similar trend.

6.5.3 Results and Discussion

Figure 6.2 presents the cosine similarity between the final model θ and the target model θ^* against various defense parameters τ. An attack is rapid if it can make $\cos(\theta, \theta^*)$ close to 1 within the fixed budget of poisoning examples. In contrast, a defense is successful if $\cos(\theta, \theta^*)$ is close to -1, i.e. θ is close to the model under no poisoning. We also show θ's error rate on clean test examples in Appendix C.2.3. Figure 6.3 presents the online classification error rate over clean examples in the stream against defense parameter τ when 10% of the stream is poisoned. A defense is effective if the error rate is low for all attack baselines.

Semi-online Experiments

Overall Performance of Attacks. Simplistic, Greedy and Semi-Online-WK are effective online attacks – $\cos(\theta, \theta^*)$ increases as τ increases and approaches 1 for large τ in most cases. Concentrated is ineffective for ignoring the online nature of the problem. Semi-Online-WK typically performs the best for its flexibility in poisoning examples selection.

Regime of Effectiveness for Defenses vs. Theories. In Figure 6.2, we highlight the easy and hard regime predicted by our analysis in blue and red background. The boundaries are calculated based on the initialization data set. The results corroborate our theories in the following aspects. First, poisoning is more rapid as τ increases, as larger τ corresponds to larger feasible set. Second, we see from the figure that for most datasets and defences, there are regimes where SimplisticAttack is rapid, and regimes where no attacks work – e.g. easy and hard. However, their boundaries do not always completely agree with the theoretically predicted boundaries – as is to be expected. This is because the multiplicative factor C in the lowerbound of poisoning speed implicitly depends on τ; smaller τ leads to larger C, and thus the number of poisoning examples required increases.

We show in an additional experiment in Appendix C.2.3 that the attacker can still make $\cos(\theta, \theta^*)$ close to 1 for small τ in the predicted easy regime under a larger learning rate $\eta = 1$, which lowers C.

Comparison between Defenses. L_2-norm defense is the weakest as it is always in easy regime. Slab, in contrast, is potentially the strongest defense mechanism because it has the widest hard regime – the defense is in hard regime even when the feasible set keeps 70% to 80% of clean examples.

Fully-online Experiments

Overall Effectiveness of Defenses. On all datasets, Slab defense is the most effective because it can slow down poisoning while still keeping most clean examples. L_2-norm is the least effective because it always permits the attacker to move the online model towards θ^*. L_2-distance-to-centroid is more effective than L_2-norm on BreastCancer, MNIST and FashionMNIST but not on IMDB.

Online Error Rate across Data Sets and its Implication. Over all defense methods, IMDB has the largest online classification error rate and BreastCancer has the lowest. MNIST and FashionMNIST have similar behaviors. The results corroborate our intuition on the impact of the learning task's difficulty level. BreastCancer is an easy low dimensional learning task, so poisoning is quickly negated by learning on clean examples. IMDB is a harder task with high dimensional inputs; poisoning is more rapid than learning for weak defenses and the attack can cause more online classifcation error than the number of poisoning examples. MNIST/FashionMNIST are high dimensional but the inputs are more structured, e.g. all centered and similar in size, therefore the results are in between.

Effectiveness of Defenses vs. Choice of Defense Parameter. The lowest online classification error rate mostly occurs when τ is set to include 70% or 90% of the examples in the stream. Defenses with too small τ filter out too many clean example and

slows down learning, while too large τ allows fast poisoning. The result suggests that a moderate τ that keeps the majority of clean examples is good in practice.

6.6 Conclusion

In conclusion, we perform a thorough theoretical and experimental study of defenses against data poisoning in online learning. In semi-online setting, we show different regimes of effectiveness for typical defenses in theory, and validate the predicted effectiveness in experiment. In fully-online setting, we show by example that poisoning effect also depends on the difficulty of the learning task, and validate the finding on real-world data sets. Our experiment suggests that the Slab defense is often highly effective in practice. For future work, we want to extend the analysis to more complex models such as neural nets, to more attack objectives such as targeted attack at specific test instance, clean-label stealthy attack and backdoor attack, and provide defenses with provable guarantees in each case.

6.7 Acknowledgement

Chapter 6 is based on the material submitted to International Conference of Machine Learning 2020 (Yizhen Wang, Somesh Jha and Kamalika Chaudhuri, "An Investigation of Data Poisoning Defenses for Online Learning"). The dissertation author is the primary investigator and co-author of this material.

Appendix A

Additional Proof and Experiment Procedure for Chapter 3

A.1 Proofs from Section 3.3

A.1.1 Proofs for Constant k

Proof. (Of Theorem 3.3.1) To show convergence in probability, we need to show that for all $\epsilon, \delta > 0$, there exists an $n(\epsilon, \delta)$ such that $\Pr(\rho(A_k(S_n, \cdot), x) \geq \epsilon) \leq \delta$ for $n \geq n_0(\epsilon, \delta)$.

The proof will again proceed in two stages. First, we show in Lemma A.1.1 that if the conditions in the statement of Theorem 3.3.1 hold, then there exists some $n(\epsilon, \delta)$ such that for $n \geq n(\epsilon, \delta)$, with probability at least $1 - \delta$, there exists two points x_+ and x_- in $B(x, \epsilon)$ such that (a) all k nearest neighbors of x_+ have label 1, (b) all k nearest neighbors of x_- have label 0, and (c) $x_+ \neq x_-$.

Next we show that if the event stated above happens, then $\rho(A_k(S_n, \cdot), x) \leq \epsilon$. This is because $A_k(S_n, x_+) = 1$ and $A_k(S_n, x_-) = 0$. No matter what $A_k(S_n, x)$ is, we can always find a point x' that lies in $\{x_+, x_-\} \subset B(x, \epsilon)$ such that the prediction at x' is different from $A_k(S_n, x)$. □

Lemma A.1.1. *If the conditions in the statement of Theorem 3.3.1 hold, then there exists some $n(\epsilon, \delta)$ such that for $n \geq n(\epsilon, \delta)$, with probability at least $1 - \delta$, there are two points x_+ and x_- in $B(x, \epsilon)$ such that (a) all k nearest neighbors of x_+ have label 1, (b) all k nearest neighbors of x_- have label 0, and (c) $x_+ \neq x_-$.*

Proof. (Of Lemma A.1.1) The proof consists of two major components. First, for large enough n, with high probability there are many disjoint balls in the neighborhood of x such that each ball contains at least k points in S_n. Second, with high probability among these balls, there exists a ball such that the k neareast neighbors of its center all have label 1. Similarly, there exists a ball such that the k nearest neighbor of its center all have label 0.

Since μ is absolutely continuous with respect to Lebesgue measure in the neighborbood of x and η is continuous, then for any $m \in \mathbb{Z}+$, we can always find m balls $B(x_1, r_1), \cdots, B(x_m, r_m)$ such that (a) all m balls are disjoint, and (b) for all $i \in \{1, \cdots, m\}$, we have $x_i \in B(x, \epsilon)$, $\mu(B(x_i, r_i)) > 0$ and $\eta(x) \in (0, 1)$ for $x \in B(x_i, r_i)$. For simplicity, we use B_i to denote $B(x_i, r_i)$ and $c_i(n)$ to denote the number of points in $B_i \bigcap S_n$. Also, let $\mu_{\min} = \min_{i \in \{1, \cdots, m\}} \mu(B_i)$. Then by Hoeffding's inequality, for each ball B_i and for any $n > \frac{k+1}{\mu_{\min}}$,

$$\Pr[c_i(n) < k] \leq \exp(-2n\mu_{\min}^2/(k+1)^2),$$

where the randomness comes from drawing sample S_n. Then taking the union bound over all m balls, we have

$$\Pr[\exists i \in \{1, \cdots, m\} \text{ such that } c_i(n) < k] \leq m \exp(-2n\mu_{\min}^2/(k+1)^2), \tag{A.1}$$

which implies that when $n > \max\left(\frac{k+1}{\mu_{\min}}, \frac{[\log m - \log(\delta/3)](k+1)^2}{\mu_{\min}^2}\right)$, with probability at least $1 - \delta/3$, each of $B_1, \cdots, B_m$ contains at least k points in S_n.

An important consequence of the above result is that with probability at least $1 - \delta/3$, the set of k nearest neighbors of each center x_i of B_i is completely different from

another center x_j's, so the labels of x_i's k nearest neighbors are independent of the labels of x_j's k nearest neighbors.

Now let $\eta_{\min,+} = \min_{x \in B_1 \bigcup \cdots \bigcup B_m} \eta(x)$ and $\eta_{\min,-} = \min_{x \in B_1 \bigcup \cdots \bigcup B_m} (1-\eta(x))$. Both $\eta_{\min,+}$ and $\eta_{\min,-}$ are greater than 0 by the construction requirements of $B_1, \cdots, B_m$. For any x_i,

$$\Pr[x_i\text{'s } k \text{ nearest neighbors all have label 1}] \geq \eta_{\min,+}^k$$

Then,

$$\Pr[\exists i \in \{1, \cdots, m\} \text{s.t. } x_i\text{'s } k \text{ nearest neighbor all have label 1}] \geq 1-(1-\eta_{\min,+}^k)^m, \quad \text{(A.2)}$$

which implies when $m \geq \frac{\log \delta/3}{\log(1-\eta_{\min,+}^k)}$, with probability at least $1 - \delta/3$, there exists an x_i s.t. its k nearest neighbors all have label 1. This x_i is our x_+.

Similarly,

$$\Pr[\exists i \in \{1, \cdots, m\} \text{ s.t. } x_i\text{'s } k \text{ nearest neighbor all have label 0}] \geq 1 - (1 - \eta_{\min,-}^k)^m, \quad \text{(A.3)}$$

and when $m \geq \frac{\log \delta/3}{\log(1-\eta_{\min,-}^k)}$, with probability at least $1 - \delta/3$, there exists an x_i s.t. its k nearest neighbors all have label 0. This x_i is our x_-.

Combining the results above, we show that for

$$n > \max\left(\frac{k+1}{\mu_{\min}}, \frac{[\log m - \log(\delta/3)](k+1)^2}{\mu_{\min}^2}\right),$$

$$m \geq \max\left(\frac{\log \delta/3}{\log(1-\eta_{\min,+}^k)}, \frac{\log \delta/3}{\log(1-\eta_{\min,-}^k)}\right),$$

with probability at least $1 - \delta$, the statement in Lemma A.1.1 is satisfied. □

A.1.2 Theorem and proof for k-nn robustness lower bound.

Theorem 3.3.1 shows that k-NN is inherently non-robust in the low k regime if $\eta(x) \in (0,1)$. On the contrary, k-NN can be robust at x if $\eta(x) \in \{0,1\}$. We define the r-robust (p,Δ)-interior as follows:

$$\begin{aligned}
\hat{\mathcal{X}}^{+}_{r,\Delta,p} &= \{x \in supp(\mu) | \forall x' \in B^o(x,r), \\
&\quad \forall x'' \in B(x', r_p(x')), \eta(x'') \geq 1/2 + \Delta\} \\
\hat{\mathcal{X}}^{-}_{r,\Delta,p} &= \{x \in supp(\mu) | \forall x' \in B^o(x,r), \\
&\quad \forall x'' \in B(x', r_p(x')), \eta(x'') \leq 1/2 - \Delta\}
\end{aligned}$$

The definition is similar to the strict r-robust (p,Δ)-interior in Section 3.4, except replacing $<$ and $>$ with $\leq$ and $\geq$. Theorem A.1.2 show that k-NN is robust at radius r in the r-robust $(1/2,p)$-interior with high high probability. Corollary A.1.3 shows the finite sample rate of the robustness lowerbound.

Theorem A.1.2. *Let $x \in \mathcal{X} \cap supp(\mu)$ such that (a) μ is absolutely continuous with respect to the Lebesgue measure (b) $\eta(x) \in \{0,1\}$. Then, for fixed k, there exists an n_0 such that for $n \geq n_0$,*

$$\Pr[\rho(A_k(S_n,\cdot),x) \geq r] \geq 1 - \delta$$

for all x in $\hat{\mathcal{X}}^{+}_{r,1/2,p} \bigcup \hat{\mathcal{X}}^{-}_{r,1/2,p}$ for all $p > 0, \delta > 0$.

In addition, with probability at least $1-\delta$, the astuteness of the k-NN classifier is at least:

$$\mathbb{E}(\mathbf{1}(X \in \hat{\mathcal{X}}^{+}_{r,1/2,p} \bigcup \hat{\mathcal{X}}^{-}_{r,1/2,p}))$$

Proof. The k-NN classifier $A_k(S_n,\cdot)$ is robust at radius r at x if for every $x' \in B^o(x,r)$, a) there are k training points in $B(x', r_p(x'))$, and b) more than $\lfloor k/2 \rfloor$ of them have the same label as $A_k(S_n,x)$. Without loss of generality, we look at a point $x \in \hat{\mathcal{X}}^{+}_{r,1/2,p}$. The second

condition is satisfied since $\eta(x) = 1$ for all training points in $B(x', r_p(x'))$ by the definition of $\hat{\mathcal{X}}^+_{r,1/2,p}$.

It remains to check the first condition. Let B be a ball in $\mathbb{R}^d$ and $n(B)$ be the number of training points in B. Lemma 16 of [CD10] suggests that with probability at least $1 - \delta$, for all B in $\mathbb{R}^d$,

$$\mu(B) \geq \frac{k}{n} + \frac{C_o}{n}\left(d\log n + \log\frac{1}{\delta} + \sqrt{k\left(d\log n + \log\frac{1}{\delta}\right)}\right) \tag{A.4}$$

implies $n(B) \geq k$, where C_o is a constant term. Let $B = B(x', r_p(x'))$. By the definition of r_p, $\mu(B) \geq p > 0$. Then as $n \to \infty$, Inequality A.4 will eventually be satisfied, which implies B contains at least k training points. The first condition is then met.

The astuteness result follows because $A_k(S_n, x) = y = 1$ in $\hat{\mathcal{X}}^+_{r,1/2,p}$ and $A_k(S_n, x) = y = 0$ in $\hat{\mathcal{X}}^-_{r,1/2,p}$ with probability 1. □

Corollary A.1.3. *For* $n \geq \max(10^4, c^4_{d,k,\delta}/[(k+1)^2p^2])$ *where*

$$c_{d,k,\delta} = 4(d+1) + \sqrt{16(d+1)^2 + 8(\ln(8/\delta) + k + 1)}$$

, with probability at least $1 - 2\delta$*,* $\rho(A_k(S_n, x)) \geq r$ *for all* x *in* $\hat{\mathcal{X}}^+_{r,1/2,p} \bigcup \hat{\mathcal{X}}^-_{r,1/2,p}$ *and for all* $p > 0, \delta > 0$*.*

In addition, with probability at least $1 - 2\delta$*, the astuteness of the k-NN classifier is at least:*

$$\mathbb{E}(\mathbf{1}(X \in \hat{\mathcal{X}}^+_{r,1/2,p} \bigcup \hat{\mathcal{X}}^-_{r,1/2,p}))$$

Proof. Without loss of generality, we look at a point $x \in \hat{\mathcal{X}}^+_{r,1/2,p}$. Let $B = B(x', r_p(x'))$, $J(B) = \mathbb{E}(Y \cdot \mathbf{1}(X \in B))$ and $\hat{J}(B)$ be the empirical estimation of $J(B)$. Notice that $\hat{J}(B)n$ is the number of training points in B, because $\eta(x) = 1$ for all $x \in B$ by the definition of

r-robust $(1/2, p)$-interior. It remains to find a threshold n such that for all $n' > n$,

$$\hat{J}(B) \geq (k+1)/n' \tag{A.5}$$

By Lemma A.1.5, with probability $1 - 2\delta$,

$$\hat{J}(B) \geq p - 2\beta_n\sqrt{p} - 2\beta_n^2 \tag{A.6}$$

for all $B \in \mathbb{R}^d$. □

Therefore it suffices to find a threshold n that satisfies

$$p - 2\beta_n\sqrt{p} - 2\beta_n^2 \geq (k+1)/n, \tag{A.7}$$

where $\beta_n = \sqrt{(4/n)((d+1)\ln 2n + \ln(8/\delta))}$.

Solving this quadratic inequality yields

$$\beta_n \leq \frac{-\sqrt{p} + \sqrt{3p + (k+1)/n}}{2}, \tag{A.8}$$

which can be re-written as

$$(8/\sqrt{n})[(d+1)\ln(2n) + \ln(8/\delta) + (k+1)/8] \leq \sqrt{(k+1)p} \tag{A.9}$$

by substituting the expression for β_n. This inequality does not admit an analytic solution. Nevertheless, we observe that $n^{1/4} \geq \ln(2n)$ for all $n \geq 10^4$. Therefore it suffices to find an $n \geq 10^4$ such that

$$(8/\sqrt{n})[(d+1)n^{1/4} + \ln(8/\delta) + (k+1)/8] \leq \sqrt{(k+1)p}. \tag{A.10}$$

Let $m = n^{1/4}$. Inequality A.10 can be re-written as

$$\sqrt{(k+1)p}m^2 - 8(d+1)m - (8\ln(8/\delta) + (k+1)) \geq 0. \tag{A.11}$$

Solving this quadratic inequality with respct to m gives

$$m \geq \frac{4(d+1) + \sqrt{16(d+1)^2 + 8(\ln(8/\delta) + k + 1)}}{\sqrt{(k+1)p}}. \tag{A.12}$$

Letting

$$c_{d,k,\delta} = 4(d+1) + \sqrt{16(d+1)^2 + 8(\ln(8/\delta) + k + 1)}$$

, we find a desired threshold

$$n = \max(10^4, m^4) \geq \max(10^4, c_{d,k,\delta}^4/[(k+1)^2p^2]). \tag{A.13}$$

The astuteness result follows in a similar way to Theorem A.1.2.

A.1.3 Proofs for High k

Robustness of the Bayes Optimal Classifier

Proof. (Of Theorem 3.3.2) Suppose $x \in \mathcal{X}^+_{r,0,0}$. Then, $g(x) = 1$. Consider any $x' \in B^o(x, r)$; by definition, $\eta(x') > 1/2$, which implies that $g(x') = 1$ as well. Thus, $\rho(g, x) \geq r$. The other case ($x \in \mathcal{X}^-_{r,0,0}$) is symmetric.

Consider an $x \in \mathcal{X}^+_{r,0,0}$ (the other case is symmetric). We just showed that g has robustness radius $\geq r$ at x. Moreover, $p(y = 1 = g(x)|x) = \eta(x)$; therefore, g predicts the correct label at x with probability $\eta(x)$. The theorem follows by integrating over all x in $\mathcal{X}^+_{r,0,0} \cup \mathcal{X}^-_{r,0,0}$. □

Robustness of k-Nearest Neighbor

We begin by stating and proving a more technical version of Theorem 3.3.3.

Theorem A.1.4. *For any n and data dimension d, define:*

$$\begin{aligned} a_n &= \frac{C_0}{n}(d\log n + \log(1/\delta)) \\ b_n &= C_0\sqrt{\frac{d\log n + \log(1/\delta)}{n}} \\ \beta_n &= \sqrt{(4/n)((d+1)\ln 2n + \ln(8/\delta))} \end{aligned}$$

where C_0 is the constant in Theorem 15 of [CD10]. Now, pick k_n and Δ_n so that $\Delta_n \to 0$ and the following condition is satisfied:

$$\frac{k_n}{n} \geq \frac{2\beta_n + b_n + \sqrt{(2\beta_n + b_n)^2 + 2\Delta_n(2\beta_n^2 + a_n)}}{\Delta_n}$$

and set

$$\begin{aligned} p_n &= \frac{k_n}{n} + \frac{C_0}{n}\Big(d\log n + \log(1/\delta) \\ &\quad + \sqrt{k_n(d\log n + \log(1/\delta)}\Big) \end{aligned}$$

Then, with probability $\geq 1 - 3\delta$, k_n-NN has robustness radius r at all $x \in \mathcal{X}^+_{r,\Delta_n,p_n} \cup \mathcal{X}^-_{r,\Delta_n,p_n}$. In addition, with probability $\geq 1 - \delta$, the astuteness of k_n-NN is at least:

$$\mathbb{E}[\eta(X)\cdot\mathbf{1}(X \in \mathcal{X}^+_{r,\Delta_n,p_n})] + \mathbb{E}(1-\eta(X))\cdot\mathbf{1}(X \in \mathcal{X}^-_{r,\Delta_n,p_n})]$$

Before we prove Theorem A.1.4, we need some definitions and lemmas.

For any Euclidean ball B in $\mathbb{R}^d$, define $J(B) = \mathbb{E}[Y \cdot \mathbf{1}(X \in B)]$ and $\hat{J}(B)$ as the

corresponding empirical quantity.

Lemma A.1.5. *With probability $\geq 1 - 2\delta$, for all balls B in $\mathbb{R}^d$, we have:*

$$|J(B) - \hat{J}(B)| \leq 2\beta_n^2 + 2\beta_n \min(\sqrt{J(B)}, \sqrt{\hat{J}(B)}),$$

where $\beta_n = \sqrt{(4/n)((d+1)\ln 2n + \ln(8/\delta))}$.

Proof. (Of Lemma A.1.5) Consider the two functions: $h_B^+(x,y) = \mathbf{1}(y = 1, x \in B)$ and $h_B^-(x,y) = \mathbf{1}(y = -1, x \in B)$. From Lemma A.1.6, both h_B^+ and h_B^- are 0/1 functions with VC dimension at most $d+1$. Additionally, $J(B) = \mathbb{E}[h_B^+] - \mathbb{E}[h_B^-]$. Applying Theorem 15 of [CD10], along with an union bound gives the lemma. □

Lemma A.1.6. *For an Euclidean ball B in $\mathbb{R}^d$, define the function $h_B^+ : \mathbb{R}^d \times \{-1, +1\} \to \{0, 1\}$ as:*

$$h_B^+(x,y) = 1(y = 1, x \in B)$$

and let $\mathcal{H}_B = \{h_B^+\}$ be the class of all such functions. Then the VC-dimension of $\mathcal{H}_B$ is at most $d+1$.

Proof. (Of Lemma A.1.6) Let U be a set of $d+2$ points in $\mathbb{R}^d$; as the VC dimension of balls in $\mathbb{R}^d$ is $d+1$, U cannot be shattered by balls in $\mathbb{R}^d$. Let $U_L = \{(x,y)|x \in U\}$ be a labeling of U that cannot be achieved by any ball (with pluses inside and minuses outside); the corresponding $d+1$-dimensional points cannot be labeled accordingly by h_B^+. Since U is an arbitrary set of $d+2$ points, this implies that any set of $d+2$ points in $\mathbb{R}^d \times \{-1, +1\}$ cannot be shattered by $\mathcal{H}_B$. The lemma follows. □

Lemma A.1.7. *Let $\delta_p = \frac{C_0}{n}\left(d\log n + \log(1/\delta) + \sqrt{k(d\log n + \log(1/\delta)}\right)$. Then, with probability $\geq 1-\delta$, for all x, $\|x - X_{(k+1)}(x)\| \leq r_{k/n+\delta_p}(x)$, and $\mu(B(x, \|x - X_{(k+1)}(x)\|)) \geq \frac{k}{n} - \delta_p$.*

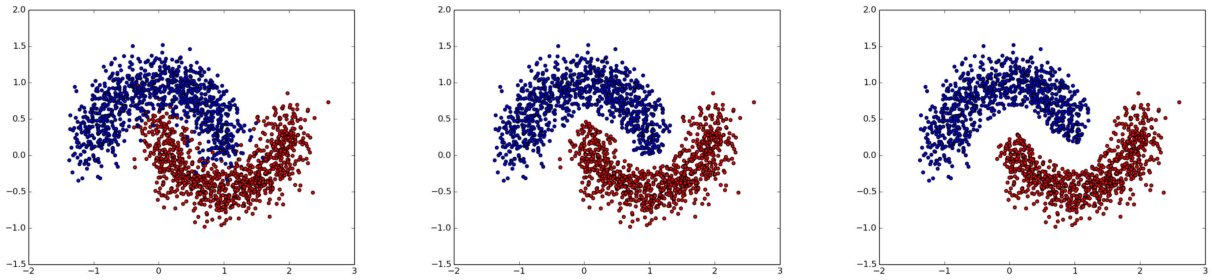

Figure. A.1: Visualization of the halfmoon dataset. 1) Training sample of size $n = 2000$, 2) subset selected by Robust_1NN with defense radius $r = 0.1$, 3) subset selected by Robust_1NN with defense radius $r = 0.2$.

Proof. (Of Lemma A.1.7) Observe that by definition for any x, r_p is the smallest r such that $\mu(B(x, r_p(x)) \geq p$. The rest of the proof follows from Lemma 16 of [CD10]. □

Proof. (Of Theorem A.1.4)

From Lemma A.1.7, by uniform convergence of $\hat{\mu}$, with probability $\geq 1 - \delta$, for all x', $\|x' - X^{(k_n)}(x')\| \leq r_{p_n}(x')$ and $\mu(B(x, \|x - X^{(k_n)}(x)\|)) \geq \frac{k_n}{n} - \delta_p$. If $x' \in \mathcal{X}^{+}_{r,\Delta_n,p_n}$, this implies that for all $\tilde{x} \in B(x', X^{(k_n)}(x'))$, $\eta(\tilde{x}) \geq 1/2 + \Delta$. Therefore, for such an x', $J(B(x', X^{(k_n)}(x'))) \geq (\frac{1}{2} + \Delta_n)\mu(B(x', X^{(k_n)}(x'))) \geq (\frac{1}{2} + \Delta_n)(k_n/n - \delta_p)$. Since for $B(x', X^{(k_n)}(x'))$, $\hat{\mu}(B(x', X^{(k_n)}(x'))) = \frac{k_n}{n}$, $\min(\hat{J}, J) \leq \frac{k}{n}$. Thus we can apply Lemma A.1.5 to conclude that

$$\hat{J}(B) > J(B) - 2\beta_n^2 - 2\beta_n\sqrt{k_n/n} > \frac{k_n}{2n},$$

which implies that $\hat{Y}(B) = \frac{1}{k_n}\sum_{i=1}^{k_n} Y^{(i)}(x) = \frac{n}{k_n}\hat{J}(B) > \frac{1}{2}$. The first part of the theorem follows.

For the second part, observe that for an $x \in \mathcal{X}^{+}_{r,\Delta_n,p_n}$, the label Y is equal to $+1$ with probability $\eta(x)$ and for an $x \in \mathcal{X}^{-}_{r,\Delta_n,p_n}$, the label Y is equal to -1 with probability $1 - \eta(x)$. Combining this with the first part completes the proof. □

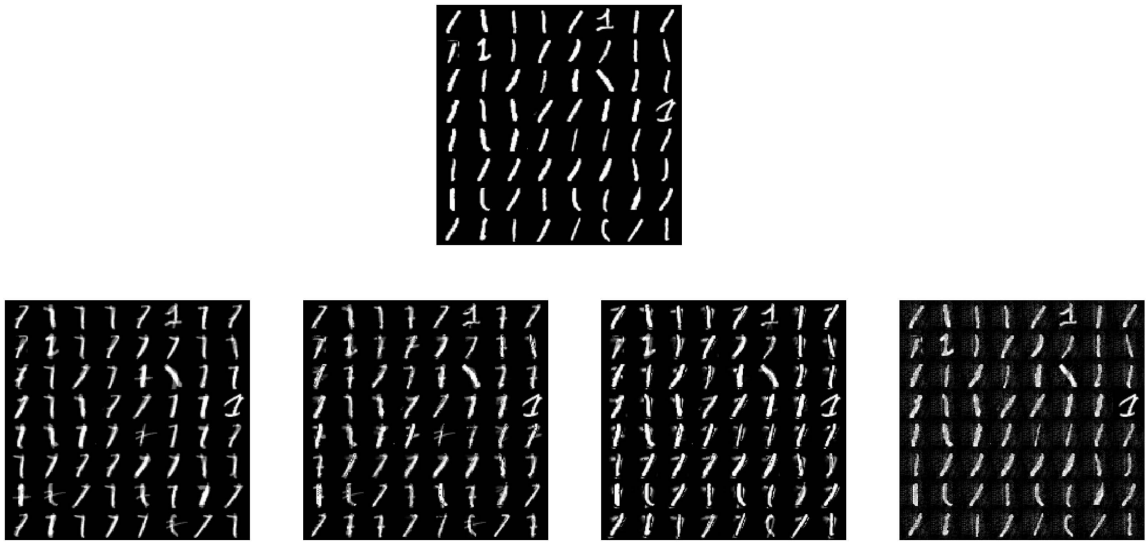

Figure. A.2: Adversarial examples of MNIST digit 1 images created by different attack methods. *Top row:* clean digit 1 test images. *Middle row from left to right:* 1) direct attack, 2) white-box kernel attack. *Bottom row from left to right:* 1) black-box kernel attack, 2) black-box neural net substitute attack.

A.2 Proofs from Section 3.4

We begin with a statement of Chernoff Bounds that we use in our calculations.

Theorem A.2.1. *[MU05] Let X_i be a 0/1 random variable and let $X = \frac{1}{m}\sum_{i=1}^{m} X_i$. Then,*

$$\Pr(|X - \mathbb{E}[X]| \geq \delta) \leq e^{-m\delta^2/2} + e^{-m\delta^2/3} \leq 2e^{-m\delta^2/3}$$

Lemma A.2.2. *Suppose we run Algorithm 1 with parameter r. Then, the points marked as red by the algorithm form an r-separated subset of the training set.*

Proof. Let $f(x_i)$ denote the output of Algorithm 2 on x_i. If $(x_i, 1)$ is a Red point, then $f(x_i) = 1 = f(x_j)$ for all $x_j \in B(x, r)$; therefore, $(x_j, -1)$ cannot be marked as Red by the algorithm as $f(x_j) \neq y_j$. The other case, where $(x_i, -1)$ is a Red point is similar. □

Table A.1: An evaluation of the black-box substitute classifier. Each black-box substitute is evaluated by: 1) its accuracy on the its training set, 2) its accuracy on the test set, and 3) the percentage of predictions agreeing with the target classifier on the test set. A combination of high test accuracy and consistency with the original classifier indicates the black-box model emulates the target classifier well.

Abalone

	target f	% training accuracy	% test accuracy	% test$f$ same as f
Kernel	StandardNN	100%	61.3%	72.6%
	RobustNN	100%	62.5%	90.9%
	ATNN	100%	61.4%	73.7%
	ATNN-All	100%	63.5%	73.5%
Neural Nets	StandardNN	69.1%	68.9%	68.6%
	RobustNN	87.2%	64.1%	86.9%
	ATNN	68.8%	68.4%	68.4%
	ATNN-All	66.5%	65.0%	66.6%

Halfmoon

	target f	% training accuracy	% test accuracy	% test same as f
Kernel	StandardNN	95.9%	95.6%	95.5%
	RobustNN	97.7%	94.9%	97.6%
	ATNN	96.4%	95.1%	96.0%
	ATNN-All	97.6%	96.8%	97.3%
Neural Nets	StandardNN	94.5%	94.0%	94.4%
	RobustNN	94.2%	90.5%	94.1%
	ATNN	95.3%	94.2%	95.2%
	ATNN-All	96.9%	96.2%	96.5%

MNIST 1v7

	target f	% training accuracy	% test accuracy	% test same as f
Kernel	StandardNN	100%	98.9%	99.3%
	RobustNN	100%	95.4%	97.6%
	ATNN	100%	98.9%	99.3%
	ATNN-All	100%	98.7%	99.3%
Neural Nets	StandardNN	99.9%	98.9%	99.1%
	RobustNN	99.8%	94.8%	98.7%
	ATNN	100%	98.8%	99.2%
	ATNN-All	99.7%	98.9%	99.3%

Lemma A.2.3. *Let $x \in \mathcal{X}$ such that Algorithm 1 finds a Red x_i within $B^o(x, \tau)$. Then, Algorithm 1 has robustness radius at least $r - 2\tau$ at x.*

Proof. For all $x' \in B(x, \tau)$, we have:

$$\|x' - x_i\| \leq \|x - x_i\| + \|x - x'\| < 2\tau$$

Since x_i is a Red point, from Lemma A.2.2, any x_j in training set output by Algorithm 1 with $y_j \neq y_i$ must have the property that $\|x_i - x_j\| > 2r$. Therefore,

$$\|x' - x_j\| \geq \|x_i - x_j\| - \|x' - x_i\| > 2r - 2\tau$$

Therefore, Algorithm 1 will assign x' the label y_i. The lemma follows. □

Lemma A.2.4. *Let B be a ball such that: (a) for all $x \in B$, $\eta(x) > \frac{1}{2} + \Delta$ and (b) $\mu(B) \geq \frac{2C_0}{n}(d \log n + \log(1/\delta))$. Then, with probability $\geq 1 - \delta$, all such balls have at least one x_i such that $x_i \in |B \cap X_n|$ and $y_i = 1$.*

Proof. Observe that $J(B) \geq \frac{C_0}{n}(d \log n + \log(1/\delta))$. Applying Theorem 16 of [CD10], this implies that $\hat{J}(B) > 0$, which gives the theorem. □

Lemma A.2.5. *Fix Δ and δ, and let $k_n = \frac{3 \log(2n/\delta)}{\Delta^2}$. Additionally, let*

$$p_n = \frac{k_n}{n} + \frac{C_0}{n}(d \log n + \log(1/\delta) + \sqrt{k_n(d \log n + \log(1/\delta))},$$

where C_0 is the constant in Theorem 15 of [CD10]. Define:

$$\begin{aligned} S_{RED} \;=\; & \{(x_i, y_i) \in S_n | x_i \in \mathcal{X}^+_{r,\Delta,p_n} \cup \mathcal{X}^-_{r,\Delta,p}, \\ & y_i = \frac{1}{2} sgn\left(\eta(x_i) - \frac{1}{2}\right) + \frac{1}{2}\} \end{aligned}$$

Then, with probability $\geq 1 - \delta$, all $(x_i, y_i) \in S_{RED}$ are marked as Red by Algorithm 1 run with parameters r, Δ and δ.

Proof. Consider a $(x_i, y_i) \in S_{RED}$ such that $x_i \in X_n \cap \mathcal{X}^+_{r,\Delta,p_n}$, and consider any $(x_j, y_j) \in S_n$ such that $x_j \in B(x_i, r)$. From Lemma A.1.7, for all such x_j, $\|x_j - X^{(k_n)}(x_j)\| \leq r_{p_n}(x_j)$; this means that all k_n-nearest neighbors x'' of such an x_j have $\eta(x'') > \frac{1}{2} + \Delta$.

Therefore, $\mathbb{E}[\sum_{l=1}^{k_n} Y^{(l)}(x_j)] \geq k_n(1/2 + \Delta)$; by Theorem A.2.1, this means that for a specific x_j, $\Pr(\sum_{l=1}^{k_n} Y^{(l)}(x_j) < 1/2) \leq 2e^{-k_n\Delta^2/3}$, which is $\leq \delta/n$ from our choice of k_n. By an union bound over all such x_j, with probability $\geq 1 - \delta$, we see that Algorithm 2 reports the label $g(x_i)$ on all such x_i, which is the same as y_i by the definition of interiors; x_i therefore gets marked as Red. □

Finally, we are ready to prove the main theorem of this section, which is a slightly more technical form of Theorem 3.4.2.

Theorem A.2.6. *Fix a Δ_n, and pick k_n and p_n as in Lemma A.2.5. Suppose we run Algorithm 1 with parameters r, Δ_n and δ. Consider the set:*

$$X_R = \left\{ x \middle| x \in \mathcal{X}^+_{r+\tau,\Delta_n,p_n} \cup \mathcal{X}^-_{r+\tau,\Delta_n,p_n}, \mu(B(x,\tau)) \geq \frac{2C_0}{n}(d\log n + \log(1/\delta)) \right\},$$

where C_0 is the constant in Theorem 15 of [CD10]. Then, with probability $\geq 1 - 2\delta$ over the training set, Algorithm 1 has robustness radius $\geq r - 2\tau$ on X_R. Additionally, its astuteness at radius $r - 2\tau$ is at least $\mathbb{E}[\eta(X) \cdot \mathbf{1}(X \in \mathcal{X}^+_{r+\tau,\Delta_n,p_n})] + \mathbb{E}[(1 - \eta(X)) \cdot \mathbf{1}(X \in \mathcal{X}^-_{r+\tau,\Delta_n,p_n})]$.

Proof. Due to the condition on $\mu(B(x,\tau))$, from Lemma A.2.4, with probability $\geq 1 - \delta$, all $x \in X_R$ have the property that there exists a (x_i, y_i) in S_n such that $y_i = g(x_i)$ and $x_i \in B(x, \tau)$. Without loss of generality, suppose that $x \in \mathcal{X}^+_{r+\tau,\Delta_n,p_n}$, so that $\eta(x) > 1/2 + \Delta_n$. Then, from the properties of r-robust interiors, this $x_i \in \mathcal{X}^+_{r,\Delta_n,p_n}$.

From Lemma A.2.5, with probability $\geq 1 - \delta$, this (x_i, y_i) is marked Red by Algorithm 1 run with parameters r, Δ_n and δ. The theorem now follows from an union

bound and Lemma A.2.3. □

A.3 Experiment Visualization and Validation

First, we show adversarial examples created by different attacks on the MNIST dataset in order to illustrate characteristics of each attack. Next, we show the subset of training points selected by Algorithm 1 on the halfmoon dataset. The visualization illustrates the intuition behind Algorithm 1 and also validates its implementation. Finally, we validate how effective the black-box subsitute classifiers emulate the target classifier.

A.3.1 Adversarial Examples Created by Different Attacks

Figure A.2 shows adversarial examples created on MNIST digit 1 images with attack radius $r = 3$. First, we observe that the perturbations added by direct attack, white-box kernel attack and black-box kernel attack are clearly targeted: either a faint horizontal stroke or a shadow of digit 7 are added to the original image. The perturbation budget is used on "key" pixels that distinguish digit 1 and digit 7, therefore the attack is effective. On the contrary, black-box attacks with neural nets substitute adds perturbation to a large number of pixels. While such perturbation often fools a neural net classifier, it is not effective against nearest neighbors. Consider a pixel that is dark in most digit 1 and digit 7 training images; adding brightness to this pixel increases the distance between the test image to training images from both classes, therefore may not change the nearest neighbor to the test image.

Figure A.2 also illustrates the break-down attack radius of visual similarity. At $r = 3$, the true class of adversarial examples created by effective attacks becomes ambiguous even to humans. Our defense is successful as the Robust_1NN classifiers still have non-trivial classification accuracy at such attack radius. Meanwhile, we should not expect robustness

against even larger attack radius since the adversarial examples at $r = 3$ are already close to the boundary of human perception.

A.3.2 Training Subset Selected by Robust_1NN

Figure A.1 shows the training set selected by Robust_1NN on a halfmoon training set of size 2000. On the original training set, we see a noisy region between the two halfmoons where both red and blue points appear. Robust_1NN cleans training points in this region so as to create a gap between the red and blue halfmoons, and the gap width increases with defense radius r.

A.3.3 Performance of Black-box Attack Substitutes

We validate the black-box substitute training process by checking the substitute's accuracy on its training set, the clean test set and the percentage of predictions agreeing with the target classifier on the clean test set. The results are shown in Table A.1. For the halfmoon and MNIST dataset, the substitute classifiers both achieve high accuracy on both the training and test sets, and are also consistent with the target classifier on the test set. The subsitututе classifiers do not emulate the target classifier on the Abalone dataset as close as on the other two datasets due to the high noise level in the Abalone dataset. Nonetheless, the substitute classifier still achieve test time accuracy comparable to the target classifier.

Appendix B

Additional Proofs and Experiment Procedure for Chapter 5

B.1 Derivation and Description of Attack Algorithms

B.1.1 Gradient Computation Using an Recurrent Prefix

Equation 5.5 shows the formula of calculating the exact online attack gradient for a general smooth function $F(\theta_t)$. In this section, we show how this gradient can be efficiently w.r.t. all points in the data stream using a recurrent prefix.

The computation for the case of $i \geq t$ and $i = t-1$ in Equation 5.5 is straightforward once $F(\theta_t)$ is known, so we focus at computing the gradient for all $i < t-1$. We call ξ_i the prefix of $\partial F(\theta_t)/\partial x_i$, where

$$\xi_i = \frac{\partial F(\theta_t)}{\partial \theta_t} \frac{\partial \theta_t}{\partial \theta_{t-1}} \cdots \frac{\partial \theta_{i+2}}{\partial \theta_{i+1}} \tag{B.1}$$

and for all $i < t-1$, we have

$$\frac{\partial F(\theta_t)}{\partial x_i} = \xi_i \frac{\partial \theta_{i+1}}{\partial x_i}. \tag{B.2}$$

Notice that the prefix has a simple recurrence relation

$$\xi_i = \xi_{i+1} \frac{\partial \theta_{i+2}}{\partial \theta_{i+1}}, \tag{B.3}$$

so it is more efficient to first find the prefix ξ_i for all $i < t-1$ and then find $\partial F(\theta_t)/\partial x_i$ from ξ_i than to compute each $\partial F(\theta_t)/\partial x_i$ from scratch. Notice that $\partial F(\theta_t)/\partial \theta_t$ and each ξ_i is a vector of dimension d. Finding $\frac{\partial \theta_{i+2}}{\partial \theta_{i+1}}$ and $\frac{\partial \theta_{i+1}}{\partial x_i}$ for a particular i takes $O(d^2)$ time each, and computing ξ_i for all i involves making $O(T)$ vector-matrix multiplications, each taking $O(d^2)$ time. Therefore the computation complexity of finding all gradients is $O(Td^2)$.

B.1.2 Gradient Computation for Online Logistic Regression

We now instantiate the gradient computation with the objective function for online logistic regression using OGD. Recall that the attacker's objective $L(\theta_t)$ at time t is the negative logistic loss of the classifier with weight θ_t over a label-flipped validation set $S_{validinv}$, i.e.

$$L(\theta_t) = \sum_{(x,y) \in S_{validinv}} -\log(1 + \exp(-y\theta_t^T x)). \tag{B.4}$$

The gradient calculation procedure in Equation 5.5 requires $\frac{\partial L(\theta_t)}{\partial \theta_t}$, $\frac{\partial \theta_{s+1}}{\partial \theta_s}$ for all $1 \leq s < t$ and $\frac{\partial \theta_{i+1}}{\partial x_i}$ for all $0 \leq i < t$, which can be found as follows:

$$\frac{\partial L(\theta_t)}{\partial \theta_t} = \sum_{(x,y) \in S_{validinv}} \frac{yx}{1 + \exp(y\theta_t^T x)}, \tag{B.5}$$

$$\frac{\partial \theta_{s+1}}{\partial \theta_s} = (1 - 2\lambda\eta_s)\mathbf{I} - \eta_s \left(\frac{\exp(y_s \theta_s^T x_s)}{[1 + \exp(y_s \theta_s^T x_s)]^2} \right) x_s x_s^T, \tag{B.6}$$

and

$$\frac{\partial \theta_{i+1}}{\partial x_i} = -\eta_i \left(\frac{-y_i \mathbf{I}}{1 + \exp(y_i \theta_i^T x_i)} + \frac{\exp(y_i \theta_i^T x_i)}{[1 + \exp(y_i \theta_i^T x_i)]^2} x_i \theta_i^T \right), \tag{B.7}$$

where λ is the regularization constant of OGD and η_i is the learning rate of OGD at time step i.

B.1.3 Attack Algorithms Descptions and Pseudocodes.

In order to simplify the algorithm pseudocode, we introduce the following notations.

First, we denote the attacker's objective function over a training stream S as $L(S)$. Recall that in Section B.1.2, we use $L(\theta_t)$ to represent the attacker's objective function at time t, where θ_t is the model parameter of the online logistic regression classifier at time t trained on stream S. Then we have $L(S) = L(\theta_T)$ for the semi-online setting and $L(S) = \sum_{t=1}^{T} L(\theta_t)$ for the fully-online setting.

Second, we use $S_{a:b}$ to represent the substream of S from S_a to S_{b-1}, and use $S_{a:b:i}$ to represent the substream containing every i-th point in S from S_a to S_{b-1}. If a substream can be represented by concatenating two above mentioned types of substreams, then its subscript will be a list of its component substream's subscript separated by comma. For example, $S_{a:b,b:c:i}$ means a substream with points from S_a to S_{b-1} and then every i-th point from S_b to S_{c-1}.

Lastly, we use ϵ_0 to denote the initial step size of gradient ascent and $\epsilon_{n_{iter}}$ to denote the gradient ascent step size at step n_{iter}. The step size $\epsilon_{n_{iter}}$ is a function of ϵ_0 and n_{iter}, which decays at the rate of $O(1/\sqrt{n_{iter}})$.

Algorithm 6 Incremental Attack(S, K, ϵ_0, max_{iter})

S: the original training stream, ϵ_0**:** initial gradient ascent step size, **K:** the attacker's budget, $\mathbf{max_{iter}}$: maximum number of iterations for gradient ascent.

$n_{iter} = 0$
$T = |S|$
$hasPerturbed = [0] \times T$
$n_{perturbed} = 0$
while ($n_{iter} < max_{iter}$) and ($n_{perturbed} <= K$) **do**
  $x_i = \arg\max_{x_k \in S} ||\frac{\partial L(S)}{\partial x_k}||_2$
  $x_i = \text{proj}_{\mathcal{F}}(x_i + \epsilon_{n_{iter}} \frac{\partial L(S)}{\partial x_i})$
  $hasPerturbed[i] = 1$
  $n_{iter} = n_{iter} + 1$
  $n_{perturbed} = sum(hasPerturbed)$
end while
return S

Algorithm 7 Interval Attack(S, K, ϵ_0, max_{iter}, s)

S: the original training stream, ϵ_0**:** initial gradient ascent step size, **K:** the attacker's budget, which is also the length of the sliding window, $\mathbf{max_{iter}}$: maximum number of iterations for gradient ascent, **s:** the step size for grid searching the best attack window.

$T = |S|$
$S_{att} = S$
for $t \in$ range $(0, T - l, s)$ **do**
  $S' = S$
  $S'_{t:t+K}$ = Find-Best-Modification-Over-Substream(S', ϵ_0, max_{iter}, $S'_{t:t+K}$)
  if $L(S') > L(S_{att})$ **then**
    $S_{att} = S'$
  end if
end for
return S_{att}

Algorithm 8 Teach-and-Reinforce Attack(S, K, ϵ_0, max_{iter}, α)

S: the original training stream, ϵ_0**:** initial gradient ascent step size, **K:** the attacker's budget, **max_iter:** maximum number of iterations for gradient ascent, α**:** fraction of attack budget used for teaching.

$T = |S|$
$s = \lceil \frac{T - \alpha K}{(1-\alpha)K} \rceil$
$S_{0:\alpha K, \alpha K:T:s}$ = Find-Best-Modification-Over-Substream(S, ϵ_0, max_{iter}, $S_{0:\alpha K, \alpha K:T:s}$)
return S

Algorithm 9 Find-Best-Modification-Over-Substream(S, ϵ_0, max_{iter}, S_{modify})

S: the original training stream, ϵ_0**:** initial gradient ascent step size, **max_iter:** maximum number of iterations for gradient ascent, $\mathbf{S_{modify}}$**:** the substream to be modified to optimize the attacker's objective.

$n_{iter} = 0$
while $n_{iter} < max_{iter}$ **do**
  for $x \in S_{modify}$ **do**
    $x = \text{proj}_{\mathcal{F}}(x + \epsilon_{n_{iter}} \partial L(S)/\partial x)$
  end for
  $n_{iter} = n_{iter} + 1$
end while
return S_{modify}

B.2 Additional Experiment Description and Result Plots

B.2.1 Dataset Size and Initial Learning Rate of Online Learners

For each data set, we select a training set of size 400 which is presented to the learner as the input stream, and a separate test set which is used for evaluation. The attacker also has a held-out validation set of size 200, and the learner has a separate held-out set of the same size as the training set, which is used to initialize the classifier w_0.

For 2D-Gaussian mixtures, MNIST 1 v.s. 7 and fashion MNIST sandals v.s. boots datasets, the initial learning rate of online learner is $\eta_0 = 0.005$ for Constant learning rate and $\eta_0 = 0.1$ for Slow Decay and Fast Decay. For UCI Spamset, the initial learning rate is $\eta_0 = 0.01$ for Constant, $\eta_0 = 0.1$ for Slow Decay and $\eta_0 = 0.2$ for Fast Decay. These parameter ensure that in all cases, the online learner has more than 90% test accuracy when run on the clean data stream.

B.2.2 Efficacy of Attack Plots

Only the result of attacks against Slow Decay online learner is presented in the main text due to the page limit. In this section, we show the plots of test accuracy against fraction of points modified for all datasets and all online learner learning rates.

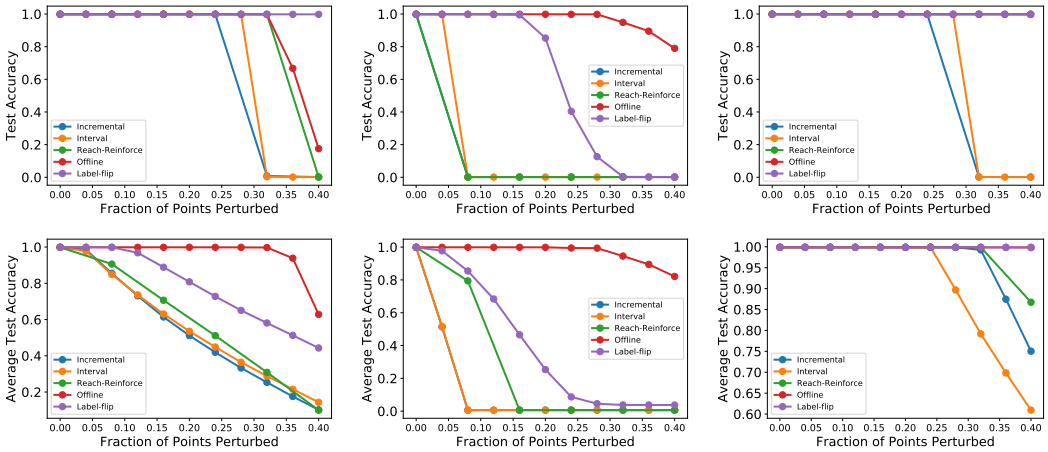

Figure. B.1: Results for 2d Gaussian mixtures. **Top row:** Semi-online, **Bottom row:** Fully-online. **Left to right:** Slow Decay, Fast Decay, Constant

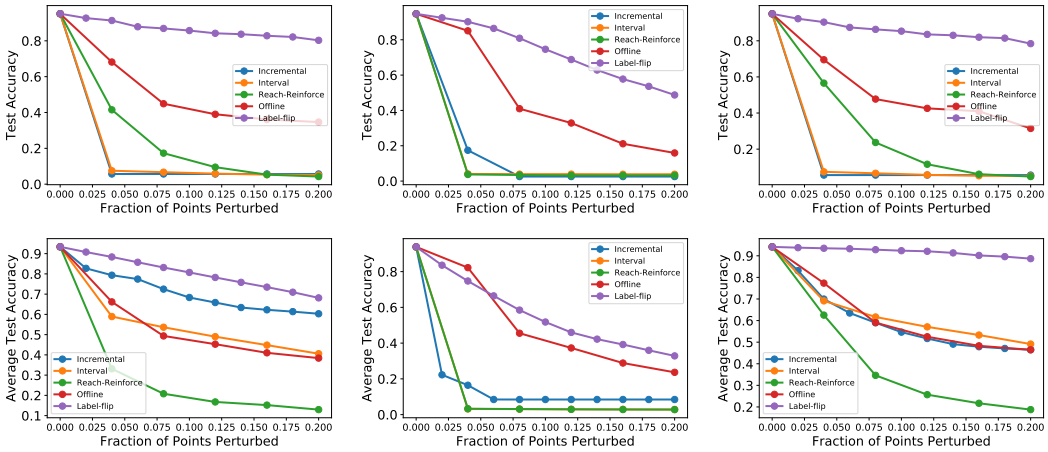

Figure. B.2: Results for MNIST. **Top row:** Semi-online, **Bottom row:** Fully-online. **Left to right:** Slow Decay, Fast Decay, Constant

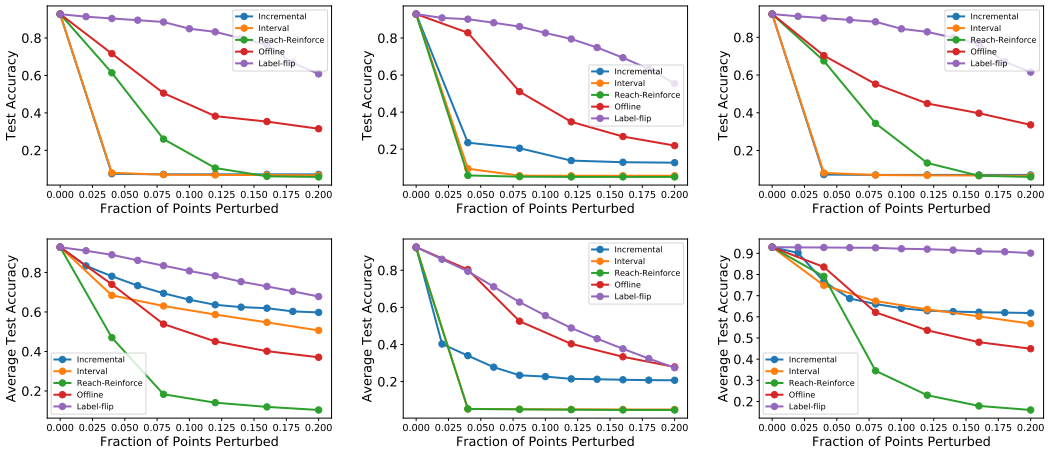

Figure. B.3: Fashion MNIST Sandals v.s. Boots. **Top row:** Semi-online, **Bottom row:** Fully-online. **Left to right:** Slow Decay, Fast Decay, Constant

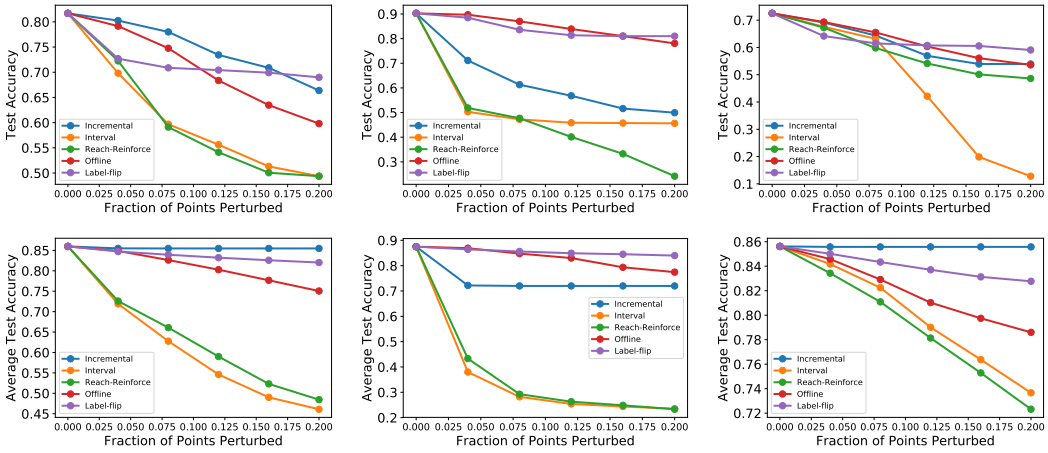

Figure. B.4: UCI Spambase. **Top row:** Semi-online, **Bottom row:** Fully-online. **Left to right:** Slow Decay, Fast Decay, Constant

B.2.3 Distribution of Attack Positions

In Table 5.1 in the experiment section, we qualitatively summarize the most frequent attack positions chosen by each online attack. In this section, we show the histogram plots of attack positions chosen by the Incremental and Interval attacks.

The entire length T window is divided into 20 equal-size bins. Each bar represents the frequency of modified points lying in the bin, normalized by the total number of points attacked in all repeated experiment. For Incremental attack, all modified points before

reaching maximum attack budget of each series or maximum number of iteration are counted. For interval attack, we count the modified points when the attack budget is 160 for 2-D Gaussian mixtures, 40 for MNIST and fashion MNIST, and 80 for UCI Spamset.

2D Gaussian Mixtures

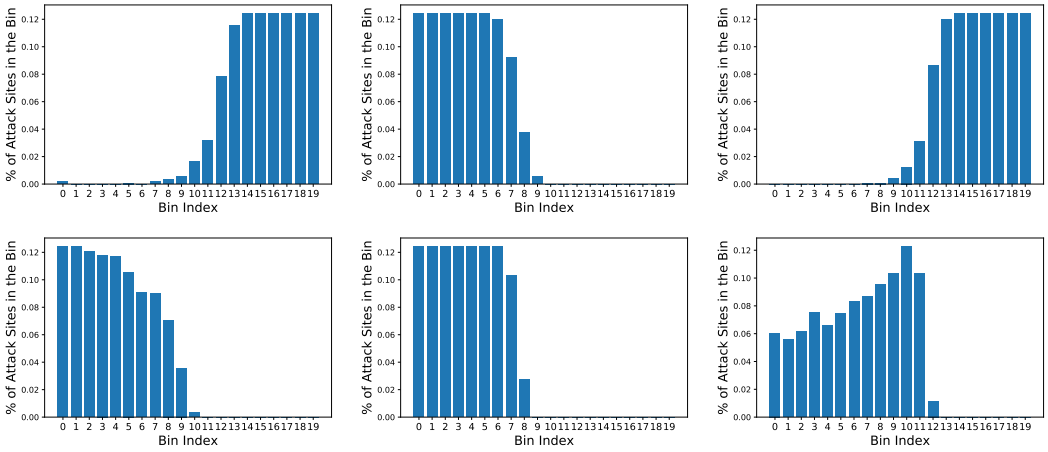

Figure. B.5: Results for 2d Gaussian mixtures, Incremental **Top row:** Semi-online, **Bottom row:** Full-online. **Left to right:** Slow Decay, Fast Decay, Constant

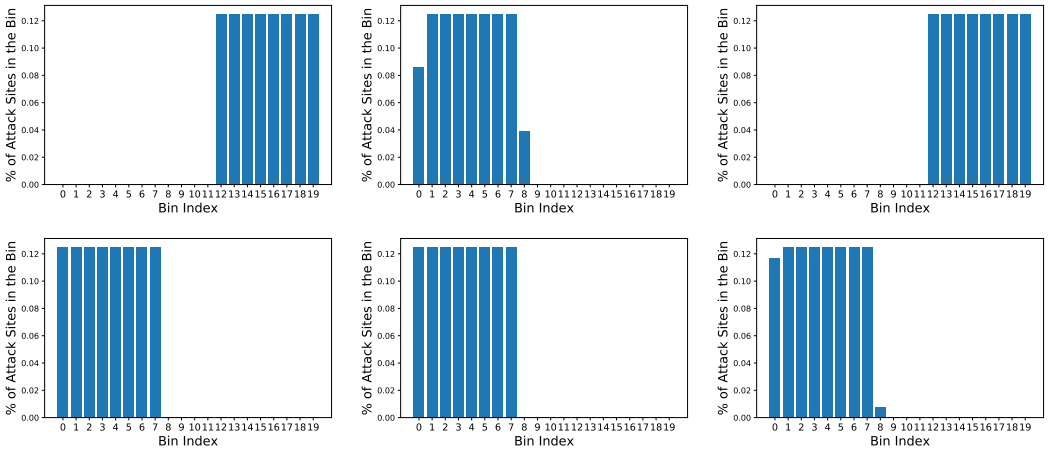

Figure. B.6: Results for 2d Gaussian mixtures, Interval **Top row:** Semi-online, **Bottom row:** Full-online. **Left to right:** Slow Decay, Fast Decay, Constant

MNIST 1 v.s. 7

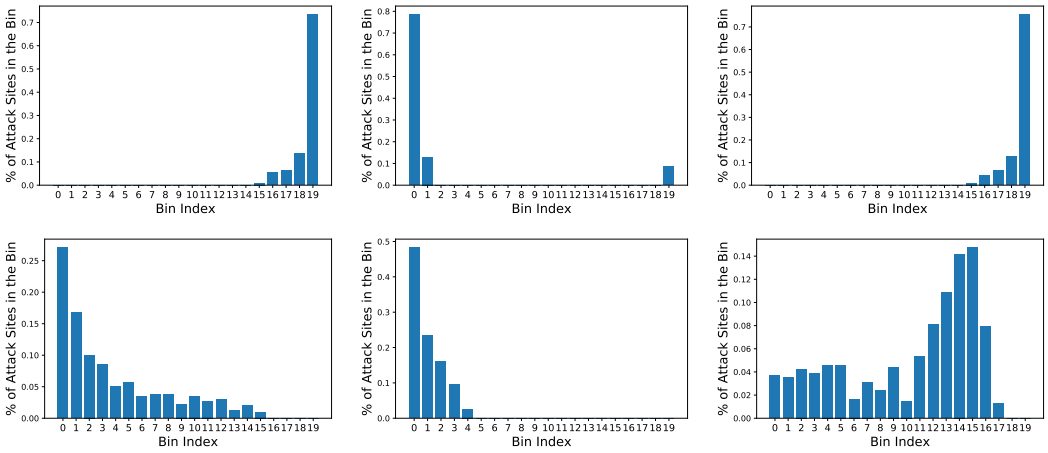

Figure. B.7: Results for MNIST, Incremental. **Top row:** Semi-online, **Bottom row:** Full-online. **Left to right:** Slow Decay, Fast Decay, Constant

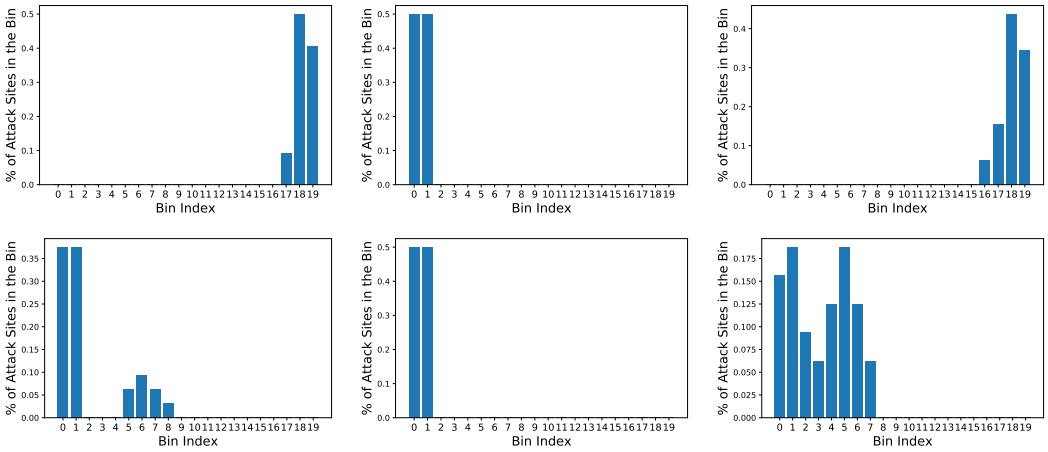

Figure. B.8: Results for MNIST, Interval **Top row:** Semi-online, **Bottom row:** Full-online. **Left to right:** Slow Decay, Fast Decay, Constant

Fashion MNIST Sandals v.s. Boots

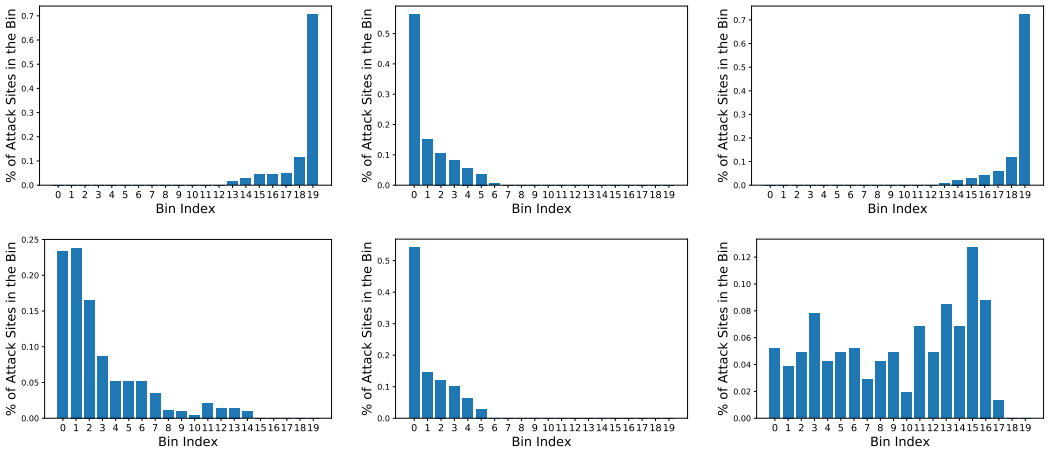

Figure. B.9: Fashion MNIST Sandals v.s. Boots, Incremental **Top row:** Semi-online, **Bottom row:** Full-online. **Left to right:** Slow Decay, Fast Decay, Constant

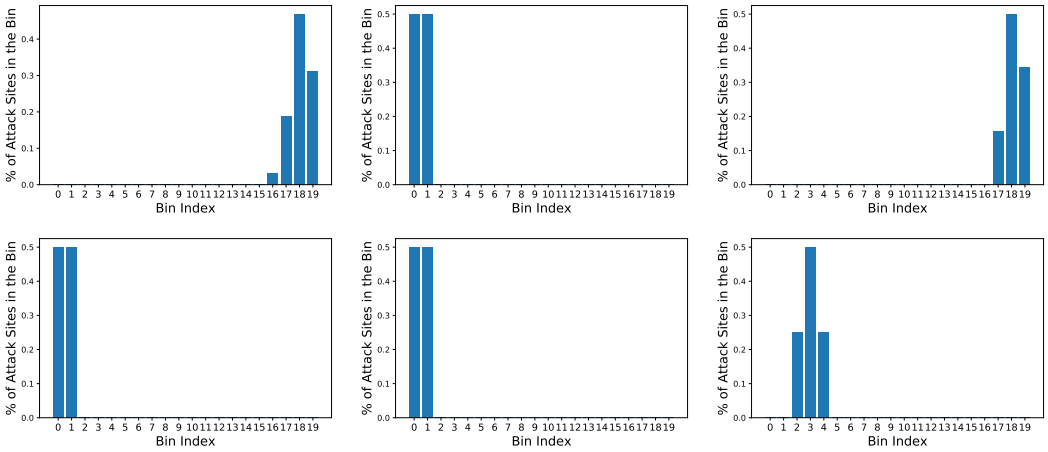

Figure. B.10: Fashion MNIST Sandals v.s. Boots, Interval **Top row:** Semi-online, **Bottom row:** Full-online. **Left to right:** Slow Decay, Fast Decay, Constant

UCI Spambase

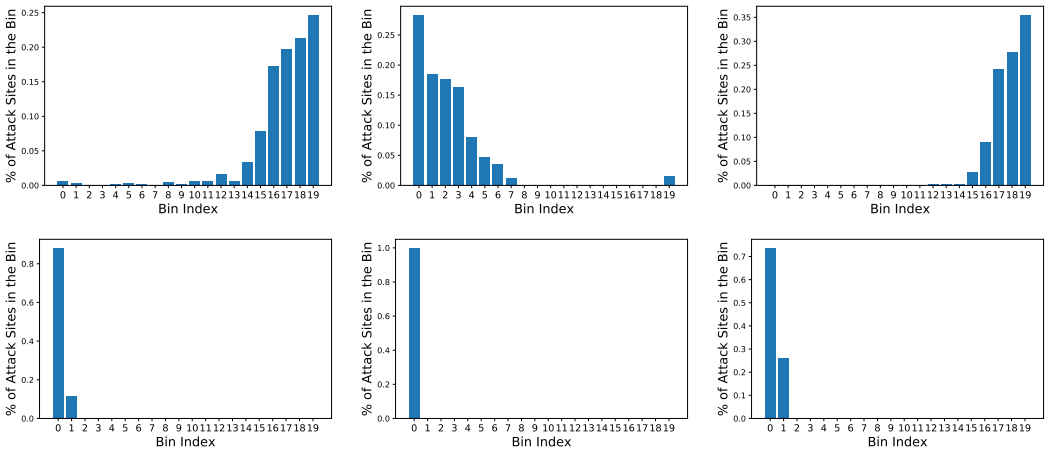

Figure. B.11: UCI Spam Dataset, Incremental **Top row:** Semi-online, **Bottom row:** Full-online. **Left to right:** Slow Decay, Fast Decay, Constant

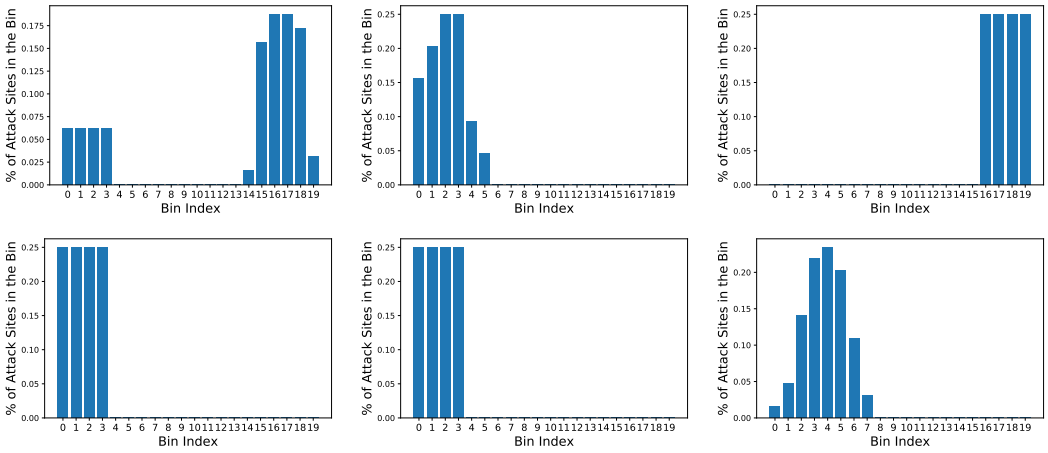

Figure. B.12: UCI Spam Dataset, Interval **Top row:** Semi-online, **Bottom row:** Full-online. **Left to right:** Slow Decay, Fast Decay, Constant

Appendix C

Additional Proof and Experiment Procedure for Chapter 6

C.1 Proofs to Theorems and Lemmas

C.1.1 Proof to Theorem 6.3.2

Recall that $\gamma_0 = \min\left(\frac{R}{\|\widetilde{\theta}_0 - \theta^*\|}, 1/\eta\right)$. In round t, we pick $\gamma_t = \min(\gamma_t^*, \gamma_0)$, where γ_t^* is the solution to $\frac{\gamma}{1+\exp(\widetilde{\theta}_t^\top(\theta^* - \widetilde{\theta}_t)\gamma)} = \frac{1}{\eta}$. Suppose we set $y_t = 1$ and $x_t = \gamma_t(\theta^* - \widetilde{\theta}_t)$; this (x, y) lies in $\mathcal{F}$ from the way we choose γ_t. Notice that $\widetilde{\theta}_t^\top \theta^* - \left\|\widetilde{\theta}_t\right\|^2 \leq \lambda^2$. If $\gamma_t = \gamma_t^*$, then the poisoner can output θ^* in the next step. Otherwise if $\gamma_t = \gamma_0$, the norm of $\widetilde{\theta}_t - \theta^*$ shrinks geometrically because

$$
\begin{aligned}
& \left\|\widetilde{\theta}_{t+1} - \theta^*\right\| \\
= & \left\|\widetilde{\theta}_t - \theta^* - \eta \frac{\gamma_0(\theta^* - \widetilde{\theta}_t)}{1 + \exp\left(\widetilde{\theta}_t^\top(\theta^* - \widetilde{\theta}_t)\gamma_0\right)}\right\| \\
= & \left\|(\widetilde{\theta}_t - \theta^*)\left(1 - \frac{\eta\gamma_0}{1 + \exp\left(\widetilde{\theta}_t^\top(\theta^* - \widetilde{\theta}_t)\gamma_0\right)}\right)\right\| \\
\leq & \left\|(\widetilde{\theta}_t - \theta^*)\left(1 - \frac{\eta\gamma_0}{1 + \exp(\lambda^2\gamma_0)}\right)\right\| \\
= & \left(1 - \frac{\eta\gamma_0}{1 + \exp(\lambda^2\gamma_0)}\right)\left\|\widetilde{\theta}_t - \theta^*\right\|.
\end{aligned}
\tag{C.1}
$$

The first inequality holds because $\widetilde{\theta}_t^\top \theta^* - \left\|\widetilde{\theta}_t\right\|^2 \leq \lambda^2$ and the monotonicity of exponential function. The last equality holds because $1 - \frac{\eta\gamma_0}{1+\exp(\lambda^2\gamma_0)} \geq 1 - \frac{\eta\gamma_0}{1+\exp\left(\widetilde{\theta}_t^\top(\theta^* - \widetilde{\theta}_t)\gamma_0\right)} \geq 0$ by our construction. The result follows from setting $C = \frac{1}{-\log\left(1 - \frac{\eta\gamma_0}{1+\exp(\lambda^2\gamma_0)}\right)}$.

Notice that if we let $\lambda = 1$, the Taylor expansion of C at $\gamma_0 = 0$ is approximately $2/(\eta\gamma_0)$. If we treat R as a variable, then when R is small, $\gamma_0 = R/\|\widetilde{\theta}_0 - \theta^*\|$, and therefore C is approximately $2\|\widetilde{\theta}_0 - \theta^*\|/(\eta R)$, which is in the order of $O(1/R)$.

C.1.2 Statement and Proof to Lemma C.1.1

Lemma C.1.1. *A data poisoner $DP^{\eta}(\theta_0, S, K, \mathcal{F}, \theta^*, \epsilon)$ can output a model θ if and only if there exists some data poisoner $DP^{\eta}(\theta_0, S, K, \mathcal{F}', \theta^*, \epsilon)$ that outputs θ.*

We first show that if a data poisoner can output θ using a sequence $\tilde{S} = \{(x_0, y_0), \cdots, (x_{T-1}, y_{T-1})\}$ with each point $(x_i, y_i) \in \mathcal{F}$, then there must be a data poisoner that output θ using a sequence $\tilde{S}' = \{(x'_0, y'_0), \cdots, (x'_{T-1}, y'_{T-1})\}$ with each point $(x'_i, y'_i) \in \mathcal{F}'$. Notice that for an OGD algorithm running on logistic regression model, $(x, +1)$ and $(-x, -1)$ will lead to the same gradient updates on the model parameter. We construct $\tilde{S}'$ as follows. For each $(x_i, y_i) \in \tilde{S}$, if $(x_i, y_i) \in \mathcal{F}'$, then we let $(x'_i, y'_i) = (x_i, y_i)$; otherwise, we let $(x'_i, y'_i) = (-x_i, -y'_i)$. Notice by the construction rule of $\mathcal{F}'$, at least one of (x_i, y_i) and $(-x_i, -y_i)$ exists in $\mathcal{F}'$. Hence, we have obtained a data poisoner on $\mathcal{F}'$ using a poisoning sequence of same length and outputing the same model as a poisoner on $\mathcal{F}$.

The reverse direction is similar. We can use the same technique to show that whenever a poisoner can output θ using a sequence of length T with points on $\mathcal{F}'$, there is a sequence using points on $\mathcal{F}$ that achieves the same goal. Therefore, the statement is true.

C.1.3 Proof to Theorem 6.3.3

For the first part, we use the same poisoning sequence as in the proof of Theorem 6.3.2; what remains to be shown is that all (x_t, y_t) in this sequence still lie in the feasible set $\mathcal{F}$. To show this, we begin with showing that for any t, $\widetilde{\theta}_t - \theta^* = c(\widetilde{\theta}_0 - \theta^*)$ for some scalar $c \in [0, 1]$.

We prove this by induction. The base case is easy – $t = 0$, and $c = 1$. In the inductive case, observe that:

$$\widetilde{\theta}_{t+1} - \theta^* = \widetilde{\theta}_t - \theta^* - \frac{\eta\gamma_t(\widetilde{\theta}_t - \theta^*)}{1 + \exp(\widetilde{\theta}_t^\top(\widetilde{\theta}_t - \theta^*)\gamma_t)},$$

and that $(1 - \eta\gamma_t/(1 + \exp(\widetilde{\theta}_t^\top(\widetilde{\theta}_t - \theta^*)\gamma_t))) \in [0, 1]$ by construction. Thus, the inductive case follows. Now, observe that the proof of Theorem 6.3.2 still goes through, as $y = 1$ and $x = \gamma(\theta^* - \widetilde{\theta}_t) = c\gamma(\theta^* - \widetilde{\theta}_0)$ is still a feasible point to add to the teaching sequence.

For the second part, observe that if the data poisoner outputs θ^*, then $\theta^* = \theta_0 - \eta \sum_i \ell'(\theta_{i-1}, x_i, y_i) y_i x_i$ for some (x_i, y_i)'s. For the augmented feasible set $\mathcal{F}'$, $y_i = +1$ always. For logistic regression, $\ell' > 0$ always. Hence, there should exist some collection of x_i's, and some positive scalars α_i such that:

$$\sum_i \alpha_i x_i = \theta^* - \theta_0.$$

However, Lemma C.1.2 shows that such collection is impossible by letting $\mathbf{u} = \theta^* - \widetilde{\theta}$ and $\beta = 1$. Therefore, no data poisoner can output θ^*.

Lemma C.1.2. *Let $\mathcal{G}$ be a convex set, $\mathbf{u}$ be a vector and $r\mathbf{u} \notin \mathcal{G}$ for all $r \geq 0$. Then there is no set of points $\{x_1, \cdots, x_n | x_i \in \mathcal{G}, \forall i \in [n]\}$ such that*

$$\sum_i \alpha_i x_i = \beta \mathbf{u}$$

for any $\alpha_i, \beta \in \mathbb{R}^+$ and $n \in \mathbb{Z}^+$.

Proof. We prove the statement by induction. The base case $n = 1$ is true because $\mathcal{G}$ does not contain any point in the direction of $\mathbf{u}$. Suppose the statement is true for $n = k$. Assume there exists a set $S = \{x_1, x_2, \cdots, x_{k+1}\}$ such that

$$\sum_{i \in [k+1]} \alpha_i x_i = \beta \mathbf{u}. \tag{C.2}$$

Let $x' = \frac{\alpha_1}{\alpha_1 + \alpha_2} x_1 + \frac{\alpha_2}{\alpha_1 + \alpha_2} x_2$. Since both x_1 and x_2 are in $\mathcal{G}$ by our assumption, x' is also in $\mathcal{G}$ by the definition of convex set. Multiplying both side of the equation above by $\frac{1}{\alpha_1 + \alpha_2}$,

we obtain

$$\begin{aligned} & \frac{1}{\alpha_1+\alpha_2}\sum_{i=1}^{k+1}\alpha_i x_i \\ = \quad & \frac{\alpha_1}{\alpha_1+\alpha_2}x_1+\frac{\alpha_2}{\alpha_1+\alpha_2}x_2+\frac{1}{\alpha_1+\alpha_2}\sum_{i=3}^{k+1}\alpha_i x_i \\ = \quad & x'+\sum_{i=3}^{k+1}\frac{\alpha_i}{\alpha_1+\alpha_2}x_i \\ = \quad & \frac{\beta}{\alpha_1+\alpha_2}\mathbf{u}, \end{aligned} \tag{C.3}$$

The equation implies that a set $\tilde{S}=\{x'\}\bigcup S\backslash\{x_1,x_2\}$ is a set of k points in $\mathcal{G}$ such that a positive linear combination of them gives a vector in the direction of $\mathbf{u}$. However, this contradicts our inductive assumption when $n=k$. Therefore, there cannot be a set of $k+1$ points from $\mathcal{G}$ that satisfies Equation C.2. The statement we want to prove is also true for $n=k+1$. □

C.1.4 Various Attack Rates in the Intermediate Regime

In Section 6.3, we conclude that the poisoning attack's rate can vary from impossible to rapid in the intermediate regime. We show three examples in which the attack is very rapid, slow and impossible.

The Rapid Case. SimplisticAttack is shown to be effective against L_2-norm defense. Let $\tilde{S}=\{(x_0,+1),\cdots,(x_{T-1},+1)\}$ be the poisoning sequence generated by SimplisticAttack against L_2-norm defense, which only uses points with $+1$ label. Let $x_{\min}$ denote the point with the smallest L_2 norm among $x_0,\cdots,x_{T-1}$. A feasible set $\mathcal{F}=\{(rx_{x_{\min}},+1)|r\geq 1\}$ contains all points in $\tilde{S}$ but not the origin. An attacker can use $\tilde{S}$ as the poisoning sequence to approach its target as rapidly as against L_2-norm defense,

even though the feasible set $\mathcal{F}$ does not contain the origin.

The Slow Case. Notice that the poisoning points in SimplisticAttack normally have diminishing magnitudes towards the end. We now show that if the attacker is only allowed to use poisoning points with constant magnitude, then the attack can be very slow.

Suppose the feasible set only contains a single point $(x, +1)$ where $x = r\frac{\theta^* - \widetilde{\theta}_0}{\left\|\theta^* - \widetilde{\theta}_0\right\|}$. The attacker can only choose $(x, +1)$ as the poisoning point. The gradient update will be

$$\theta_{t+1} = \theta_t + \frac{\eta x}{1 + \exp(\theta_t^\top x)}$$

for all t.

We consider a 1-D example with $\widetilde{\theta}_0 = 0.5$, $\theta^* = 1$, step size $\eta = 1$ and an L_2-norm upperbound $r = 10$. If the attacker can choose any point $(x, +1)$ such that $x \leq r = 10$, the attacker can reach its objective with a single poisoning points by solving the following equation

$$1 = 0.5 + \frac{x}{1 + \exp(0.5x)},$$

which gives $x \approx 1.629$.

Now suppose the attacker is only allowed to choose $(r, +1)$ as the poisoning point. Let Δ_t denote the difference between θ_{t+1} and θ_t. We have

$$\Delta_t = \frac{x}{1 + \exp(\theta_t^\top x)} = \frac{10}{1 + \exp(10\theta_t)} \leq \frac{10}{1 + \exp(5)}.$$

The inequality is because $\theta_t \in [0.5, 1]$ as it starts from $\widetilde{\theta}_0 = 0.5$ and approaches $\theta^* = 1$. The attacker then needs at least $\lceil \frac{0.5(1+\exp(5))}{10} \rceil = 8$ poisoning points to achieve its goal. Notice that the exponential term grows much more rapidly then the denominator. If $r = 20$, the attacker will need at least $\lceil \frac{0.5(1+\exp(10))}{20} \rceil = 551$ points! In short, the attacker needs exponentially many points w.r.t. r in this case to achieve the same objective that can be

done using a single point against L_2-norm defense.

This example also naturally extends to high dimensional inputs as long as the attacker only wants to change one coordinate of the model parameter.

The Impossible Case. We consider the same $\widetilde{\theta}_0, \theta^*$ and η as in the slow case. Now the attacker can only choose $(2.5, +1)$ as the poisoning point, i.e. $r = 2.5$. After one update step, the model θ becomes

$$\theta = \widetilde{\theta} + \frac{\eta x}{1 + \exp(\widetilde{\theta}^\top x)} = 0.5 + \frac{2.5}{1 + \exp(1.25)} = 1.058.$$

Notice that the model parameter monotonically increases if the attacker adds more poisoning points of $(2.5, +1)$ into the stream. Therefore, this θ after one poisoning point is the closest the attacker can ever get to the target $\theta^* = 1$. No data poisoner can output θ^* exactly.

C.1.5 Proof to Lemma 6.3.4

For simplicity, we consider a feasible set $\mathcal{F}'^c = \{(yx, +1) | (x, y) \in \mathcal{F}^c\}$, which is the one-sided form of $\mathcal{F}^c$. As a consequence of Lemma C.1.1, defenses characterized by $\mathcal{F}^c$ and $\mathcal{F}'^c$ have the same behavior. We define $\mathcal{F}'^c_- = \{(-x, +1) | (x, -1) \in \mathcal{F}^c_-\}$. Then $\mathcal{F}'^c = \mathcal{F}^c_+ \bigcup \mathcal{F}'^c_-$. Also, let $L(r, \mathbf{u})$ denote a line segment connecting the origin and a point $r\frac{\mathbf{u}}{\|\mathbf{u}\|}$ as in Theorem 6.3.3.

The Rapid Case. For the first part, we show that there exists an $r > 0$ such that $L(r, \theta^* - \widetilde{\theta}_0) \subseteq \mathcal{F}^c_+ \bigcup \mathcal{F}'^c_-$. Suppose $\|\mu_+\| < \tau$. Let α denote the angle between μ_+ and $\theta^* - \widetilde{\theta}_0$, and let $x = l(\theta^* - \widetilde{\theta}_0) / \left\|\theta^* - \widetilde{\theta}_0\right\|$ be a point in the direction of $\theta^* - \widetilde{\theta}_0$ with norm l. A point $(x, +1)$ is in $\mathcal{F}^c_+$ if and only if it satisfies the following inequality

$$\|x - \mu_+\|^2 = l^2 - 2l \|\mu_+\| \cos\alpha + \|\mu_+\|^2 \leq \tau^2.$$

We want to find the range for l when the inequality holds. Notice that the determinant of this quadratic inequality, $\|\mu_+\|^2 (\cos\alpha)^2 - \|\mu_+\|^2 + \tau^2$, is always positive. Therefore, real solutions always exist for the lower and upper bound of l. Solving the inequality gives us

$$l \geq \|\mu_+\| \cos\alpha - \sqrt{\|\mu_+\|^2 (\cos\alpha)^2 - \|\mu_+\|^2 + \tau^2}$$

and

$$l \leq \|\mu_+\| \cos\alpha + \sqrt{\|\mu_+\|^2 (\cos\alpha)^2 - \|\mu_+\|^2 + \tau^2}.$$

It is also easy to verify that the lower bound is smaller than 0 and the upper bound is larger than 0. Let $r = \|\mu_+\| \cos\alpha + \sqrt{\|\mu_+\|^2 (\cos\alpha)^2 - \|\mu_+\|^2 + \tau^2}$. The solution implies that $L(r, \theta^* - \widetilde{\theta_0}) \subseteq \mathcal{F}_+^c$.

Similarly, when $\|\mu_-\| < \tau$, we can show that there exists an r such that $L(r, \theta^* - \widetilde{\theta_0})$ is in $\mathcal{F}_-'^c$. Therefore, when $\tau > \min(\|\mu_+\|, \|\mu_-\|)$, there exists some $r > 0$ s.t. $L(r, \theta^* - \widetilde{\theta_0}) \subseteq \mathcal{F}'^c$. The rest follows from Theorem 6.3.3.

The Impossible Case. For the second part, we show that $\mathcal{G} = \{(x, +1) | (\theta^* - \theta_0)^\top x \leq 0\}$, i.e. the negative halfspace characterized by $(\theta^* - \theta_0)$, contains both $\mathcal{F}_+^c$ and $\mathcal{F}_-'^c$ yet does not intersect $L(+\infty, \theta^* - \theta_0)$ at all. The rest follows from Theorem 6.3.3.

We first look at $\mathcal{F}_+^c$. The condition $\langle \mu_+, \theta^* - \theta_0 \rangle < 0$ implies that $\mu_+ \in \mathcal{G}$. The distance between μ_+ to the hyperplane $(\theta^* - \theta_0)^\top x = 0$ is $\|u_+\|$. Since $\|u_+\| \geq \tau$, none of the point in $\mathcal{F}_+^c$ can cross the hyperplane, and therefore $\mathcal{F}_+^c \subseteq \mathcal{G}$. The proof for showing $\mathcal{F}_-'^c$ in $\mathcal{G}$ is analogous. Therefore $\mathcal{G}$ contains both $\mathcal{F}_+^c$ and $\mathcal{F}_-'^c$. In addition, $\mathcal{G}$ does not contain any point x in $L(+\infty, \theta^* - \theta_0)$ because $x = r(\theta^* - \theta_0)/\|\theta^* - \theta_0\|$ for some $r > 0$, and $(\theta^* - \theta_0)^\top x = (\theta^* - \theta_0)^\top (r(\theta^* - \theta_0)/\|\theta^* - \theta_0\|) = r > 0$.

C.1.6 Proof to Lemma 6.3.5

For simplicity, we consider a feasible set $\mathcal{F}'^s = \{(yx, +1)|(x, y) \in \mathcal{F}^s\}$, which is the one-sided form of $\mathcal{F}^s$. As a consequence of Lemma C.1.1, defenses characterized by $\mathcal{F}^s$ and $\mathcal{F}'^s$ have the same behavior. We define $\mathcal{F}'^s_- = \{(-x, +1)|(x, -1) \in \mathcal{F}^s_-\}$. Then $\mathcal{F}'^s = \mathcal{F}^s_+ \bigcup \mathcal{F}'^s_-$.

The Rapid Case. For the first part, we show that $L((\tau - b_+)/\left\|\beta\right\|, \theta^* - \widetilde{\theta}_0)$ is in $\mathcal{F}^s_+$ or $L((\tau - b_-)/\left\|\beta\right\|, \theta^* - \widetilde{\theta}_0)$ is in $\mathcal{F}'^s_-$.

Suppose $\tau - b_+ > 0 > -\tau - b_+$. Let x be a point in $L((\tau - b_+)/\left\|\beta\right\|, \theta^* - \widetilde{\theta}_0)$. Then x can be expressed as $x = r(\theta^* - \widetilde{\theta})/\left\|\theta^* - \widetilde{\theta}\right\|$ for some $0 < r < (\tau - b_+)/\left\|\beta\right\|$. We know that

$$\beta^\top x \leq \left\|\beta\right\| r \leq \left\|\beta\right\| (\tau - b_+)/\left\|\beta\right\| = \tau - b_+,$$

therefore $\beta^\top x + b_+ \leq \tau$. On the other hand, $\beta^\top x > 0$ by our assumption that $\beta^\top(\theta^* - \widetilde{\theta}_0) > 0$, and $b_+ > -\tau$ by the condition $-\tau - b_+ < 0$. Therefore, $\beta^\top x + b_+ > b_+ > -\tau$. We can conclude that the Slab score $|\beta^\top x| \leq \tau$, and hence $L((\tau - b_+)/\left\|\beta\right\|, \theta^* - \widetilde{\theta}_0) \in \mathcal{F}^s_+$.

Similarly, we can show that $L((\tau - b_-)/\left\|\beta\right\|, \theta^* - \widetilde{\theta}_0)$ is in $\mathcal{F}'^s_-$ when $\tau - b_- > 0 > -\tau - b_-$. The rest follows from Theorem 6.3.3, and an example of such a rapid data poisoner is SimplisticAttack$(\theta_0, S, K, \mathcal{F}, \theta^*, \epsilon, r)$ in which $\mathcal{F} = \mathcal{F}'^s$ and $r = \min((\tau - b_+)/\left\|\beta\right\|, (\tau - b_-)/\left\|\beta\right\|)$.

The Impossible Case. For the second part, we show that $\mathcal{G} = \{(x, +1)|\beta^\top x \leq 0\}$, i.e. the negative halfspace characterized by β, contains both $\mathcal{F}^s_+$ and $\mathcal{F}'^s_-$ yet does not intersect $L(+\infty, \theta^* - \widetilde{\theta}_0)$ at all. The rest follows from Theorem 6.3.3.

We first look at $\mathcal{F}^s_+$ when $0 > \tau - b_+ > -\tau - b_+$. We know that the Slab score $|\beta^\top x + b_+| \leq \tau$ for all $x \in \mathcal{F}^s_+$. Combining with the condition that $\tau - b_+ < 0$, we have $\beta^\top x \leq \tau - b_+ < 0$ for all $x \in \mathcal{F}^s_+$. Therefore, $\mathcal{F}^s_+ \subseteq \mathcal{G}$. The proof for showing $\mathcal{F}'^s_-$ in $\mathcal{G}$ is analogous. Also, since $\beta^\top(\theta^* - \widetilde{\theta}_0) > 0$, we have $\beta^\top x > 0$ for all $x \in L(+\infty, \theta^* - \widetilde{\theta}_0)$.

Therefore, $\mathcal{G}$ does not intersect $L(+\infty, \theta^* - \widetilde{\theta}_0)$ at all. Theorem 6.3.3 tells that no data poisoner can output θ^* starting from $\widetilde{\theta}_0$.

If $\beta^\top(\theta^* - \theta_0) > 0$, then we can replace $\widetilde{\theta}_0$ in the above analysis with θ_0, and show that $\mathcal{G}$ does not intersect $L(+\infty, \theta^* - \theta_0)$ at all. Theorem 6.3.3 then tells that no data poisoner can output θ^*.

C.1.7 Proof to Lemma 6.4.1

In this problem setting, a linear classifier with classification rule $f(x) = \text{sgn}(\theta^\top x)$ is correct on clean examples if $\theta > 0$, and is incorrect otherwise. We show that the online learner's model θ_t is always positive (except at $t = 0$) even in the presence of the attacker, and thus the online model has no classification error on the clean examples for $t > 0$.

The update rule of a logistic regression classifier at each iteration t is

$$\theta_{t+1} = \theta_t + \eta \frac{y_t x_t}{1 + \exp(\theta_t^\top x_t y_t)}.$$

For simplicity, we assume $y_t = 1$ for all t because (x_t, y_t) and $(-x_t, -y_t)$ gives the same updates. Therefore, the clean example will be $(1, +1)$ and the only feasible poisoning example is $(-1, +1)$. Substitute $\eta = 1$ into the equation, the rule becomes

$$\theta_{t+1} = \theta_t + \frac{x_t}{1 + \exp(\theta_t^\top x_t)}.$$

Now, let S be any poisoned stream created by the attacker. We use an *interval* to denote a consecutive sequence of data in S, and extract *intervals of interests* using the following rules.

An interval of interest starts at time t if 1) $\theta_t \geq 0$ and 2) $\theta_{t+1} \leq 0$. It ends at the first time $t' > t$ when 1) $\theta_{t'} \leq 0$ and 2) $\theta_t > 0$. It also ends if $t' = T$, i.e. we already reach

the end the stream.

Notice that by our rule, we can extract multiple intervals of interests from S, and the intervals will never overlap.

We observe the following two major properties of the intervals.

Observation 1. Clean examples outside the intervals will all be correctly classified. [1]

Observation 2. There are at least as many poisoning points as clean points in each interval.

The reason for Observation 1 is straight forward as $\theta_t > 0$ for all t not in the interval.

In order to prove Observation 2, we first show a few useful claims.

Claim C.1.3. *Suppose the model $\theta_t \geq 0$ at time t, and updates over a poisoning point so that $\theta_{t+1} < 0$. Then at any t' such that $\theta_{t+1} \leq \theta_{t'} \leq 0$, a single model update over a clean point will always cause $\theta_{t'+1} > 0$.*

Proof. Deriving from the model update rule, θ_{t+1} is at least -0.5. This is done by minimizing $\theta_t + \frac{-1}{1+\exp(-\theta_t)}$ for $\theta_t \geq 0$. We again can derive that a single update over a clean point always makes $\theta_{t'+1} > 0$ from the model update rule, as $\theta_t + \frac{1}{1+\exp(\theta_t)} > 0$ for all $\theta_t \in [-0.5, 0]$. □

Claim C.1.4. *Suppose $-0.5 \leq \theta_t \leq 0$, and in the following n points, there are more clean points than poisoning points. Then there must be some $t' \in \{t+1, \cdots, t+n\}$ such that $\theta_{t'} > 0$.*

Proof. We prove by contradiction. Suppose for all $t' \in \{t+1, \cdots, t+n\}$, $\theta_{t'} \leq 0$. Then an update over a poisoning point will lower θ by at most 0.5, while an update over a clean will raise θ by at least 0.5. Since there are more clean points than poisoning points, then the

[1] There exists one corner case: the first point in S is clean and since $\theta_0 = 0$, it has the chance to be misclassified. When this case happen, the error bound on clean examples needs to add 1, which does not impact the result when the stream length T is large.

accumulated change in θ must be at least 0.5 and strictly more than 0.5 when $\theta_t = -0.5$. Therefore, $\theta_{t+n} > 0$. Contradiction. □

We then prove Observation 2 by two cases as follows.

Case 1. In each interval of interest that ends before time T, the first point at the starting time t is always a poisoning point, because $\theta_{t+1} < \theta_t$, i.e. the point moves the model into the wrong direction. Similarly, the last point at time t' will always be a clean point, because it moves the model in the right direction. Using the same argument in Claim C.1.3, model θ_{t+1} is at least -0.5. Then by Claim C.1.4, the remaining points cannot contain more clean points than poisoning points, because otherwise the model θ will be positive somewhere in the interval, contradicting to the construction of the interval. Therefore, the number of poisoning points at least equal to the number of clean points in the interval.

Case 2. In the interval of interest that ends at time T if such interval exists, the first point at the starting time t is also a poisoning point due to the same argument for the previous case. Once again by Claim C.1.4, the interval must contain no more clean points than poisoning points, otherwise the model θ will be positive at some time within the interval.

Now combining Observation 1 and 2, we know that the clean points are only classified wrongly in the interval of interests, and the number of such clean points no more than the poisoning points in the intervals. Therefore, no attacker can cause more online error on clean points than the number of poisoning points it can add.

The bound is also tight as the attacker can construct an attack such that the number of error equals to the number of poisoning points. The attack can be done by inserting one poisoning point after every clean point, starting at the beginning of the stream.

C.1.8 Proof to Lemma 6.4.2

We construct a fully-online attacker which adds *one* poisoning example after every ten clean examples, and show that such attacker can achieve the attack effectiveness stated in the lemma.

The main idea of the proof is to show that for most time t, θ_t has very few nonnegative coordinates and thus low classification accuracy on clean examples. In order to show this, we look at the sum of model parameters over all coordinates, and construct an attacker such that 1) a poisoning example can lower the sum much more rapidly than a clean example increases it when θ_t has many nonnegative coordinates, and 2) the sum is also lower bounded. If the number of nonnegative coordinates in θ_t is above a threshold for too many different t, then the lower bound will be violated and thus causes contradiction.

Let $\theta_{t,i}$ denote the i-th coordinate of the online model θ_t at iteration t. In this problem setting, the classifier predicts a clean example $(e_i, +1)$ or $(-e_i, -1)$ correctly if $\theta_{t,i} > 0$ and makes an error otherwise.

The attacker is as follows. At time t when the learner gets a poisoning examples from the attacker, the attacker first picks 1000 nonnegative coordinates of θ_t. Let $J = \{j_1, \cdots, j_{1000}\}$ denote the indices of these 1000 coordinates. The poisoning point x has $x_j = -\frac{1}{\sqrt{1000}}$ for $j \in J$ and $x_j = 0$ for other $j \in \mathbb{Z}^+, j \leq 10000$. The label $y = +1$. If fewer than 1000 coordinates are nonnegative, then the attacker returns a zero vector as the poisoning example, which causes no change to the online model.

We say the online learner makes one cycle of updates when it updates on ten clean examples followed by a poisoning examples, which is the pattern of poisoning locations in the setting. Then 1,000,000 clean examples corresponds to 100,000 complete cycles.

In order to prove the lemma, we first look at the number of nonnegative coordinates in θ_t when $t = 11k + 10$ for $k \in \mathbb{Z}^+$, i.e. the time steps when the learner gets an example from the attacker.

Claim C.1.5. *The number of nonnegative coordinates in θ_{11k+10} must be below* 1000 *in at least 60,000 cycles, i.e. for 60,000 different k.*

Proof. We prove the contrapositive statement by contradiction. Let s_t denote the sum of all coordinates of θ_t, i.e. $s_t = \sum_{j=1}^{10000} \theta_{t,j}$, and $\theta_{t,i}$ denote the i-th coordinate of the online model θ_t at iteration t. We have the following observations on $\theta_{t,i}$ and s_t.

Observation 1. We have $\theta_{t,i} \geq -\frac{1}{\sqrt{1000}}$ for all t and i. This is because $\theta_{t,i}$ decreases only when the learner updates over a poisoning example and the coordinate is nonnegative, and the poisoning example can lower $\theta_{t,i}$ by at most $\frac{1}{\sqrt{1000}}$ in one update.

Observation 2. For each update over the nonzero poisoning example, $s_{t+1} - s_t \leq -\sqrt{1000}/2$. This is because for each coordinate $j \in J$, $\theta_{t+1,j} - \theta_{t,j} \leq -1/(2\sqrt{1000})$ by our construction of the attacker, and there are a total of 1000 coordinates to be updated.

Suppose that in more than 40,000 cycles, i.e. 40,000 different k, the number of nonnegative coordiates in θ_{11k+10} is more that 1000. Then for at least 40,000 different t, $s_{t+1} - s_t \leq -\sqrt{1000}/2$. On the other hand, when the learner updates over a clean example, $s_{t+1} - s_t \leq 1/(1+\exp(\theta_t^\top x_t y_t)) \leq 0.52$ because $\theta_{t,i} \geq -\frac{1}{\sqrt{1000}}$ for all i, t and $\theta_t^\top x_t y_t \geq -\frac{1}{\sqrt{1000}}$. There are a total of 1,000,000 clean examples. We can get an upper bound of s_T as follows.

$$\begin{aligned} s_T = s_T - s_0 &= \sum_{t=0}^{T-1} s_{t+1} - s_t \\ &\leq 40000(-\sqrt{1000}/2) + 1000000(0.52) \\ &\leq -10^5. \end{aligned}$$

However, by Observation 1, $\theta_{T,i} \geq -\frac{1}{\sqrt{1000}}$ for all i,

$$s_T \geq 10000(-\frac{1}{\sqrt{1000}}) \geq -400,$$

which contradicts the lower bound. Hence, it is impossible that θ_{11k+10} has at least 1000

nonnegative coordinates for more than 40,000 cycles, and the statement is proven. □

Claim C.1.6. *At least 600,000 clean examples are classified by models with fewer than 1000 nonnegative coordinates.*

Proof. Claim C.1.5 shows that θ_{11k+10} has fewer than 1000 nonnegative coordinates in over 60,000 cycles. Notice that in each cycle, the number of nonnegative coordinates of θ_t never decreases when model updates on clean examples. Therefore, θ_{11k+i} must have fewer than 1000 nonnegative coordinates too for $i = 0, \cdots, 9$, i.e. the time steps when the model updates over clean examples. There are 10 clean examples in each cycle, and hence the statement is proven. □

The last step is to show that with high probability, at least 500,000 out of the 600,000 examples in Claim C.1.6 are misclassified using concentration bound. Let J' be the set of time step indices at which the online model has fewer than 1000 nonnegative coordinates. By Claim C.1.6, $|J'| \geq 600,000$. Let $X_t = \mathbb{1}(\theta_t$ correctly predicts the clean example $(x_t, y_t))$. For $t \in J'$, $\mathbb{E}(X_t) < 0.1$ because 1) the input x_t only has one nonzero coordinate, 2) the prediction is only correct when the nonzero coordinate of x_t corresponds to a nonnegative coordinate of θ_t, and 3) x_t is independent of θ_t. In addition, X_t is bounded in $[0, 1]$. Let $S = \sum_{t \in J'} X_t$ be the random variable of total number of correct prediction for time steps in J'. Notice that

$$\mathbb{E}[S] = \sum_{t \in J'} \mathbb{E}[X_t] < 60000$$

. Applying Hoefdding's inequality, we have

$$\begin{aligned} & \Pr[S > 100000] \\ = \quad & \Pr[S - \mathbb{E}[S] > 100000 - \mathbb{E}[S]] \\ \leq \quad & \Pr[S - \mathbb{E}[S] > 40000] \\ \leq \quad & \exp\left(-\frac{2(40000)^2}{|J'|}\right) \\ \leq \quad & \exp\left(-\frac{2(40000)^2}{600000}\right) < 0.01. \end{aligned} \tag{C.4}$$

The probability that more than 100,000 examples at times steps $t \in J'$ is classified correctly is less than 0.01. Therefore, with probability more than 0.99, at least 500,000 examples at times steps $t \in J'$ are misclassified, and hence proves the lemma.

C.2 Experiment Details

C.2.1 Data Preparation

UCI Breast Cancer. Initialization, training and test data sets have size $100, 400, 100$ respectively, and the data set is divided in random.

IMDB Review We train a Doc2Vec feature extractor on the 50000 unlabeled reviews provided in the original data set using the method in [ŘS10]. Then we convert reviews in the original training and test set into feature vector of $d = 100$ using the extractor. Initialization, training and test data sets have size $5000, 10000, 5000$ respectively. The first two data sets are drawn from the original training set in vector representation without replacement. The test set is drawn from the original test set.

MNIST 1v7. We use a subset of the MNIST handwritten digit data set containing images of Digit 1 and 7. Initialization, training and test data set have size $1000, 8000, 1000$ respectively. The first two data sets are drawn from the original MNIST training sets in random with no replacement. The test set is drawn from the original MNIST test set.

fashionMNIST Bag v.s. Sandal. We use a subset of the fashionMNIST data set containing images of bags and sandal. Initialization, training, and test data set have size $1000, 8000, 1000$ respectively. The first two data sets are drawn from the original fashionMNIST training sets in random with no replacement. The test set is drawn from the original fashionMNIST test set.

Each data set is normalized by subtracting the mean of all data points and then scaled into $[-1, 1]^d$.

C.2.2 Detailed Experiment Procedures.

The learning rate η is set to 0.05 for UCI Breast Cancer and 0.01 for the other three data sets. The model obtained after making one pass of the training set using OGD

algorithm has high accuracy on all data sets.

For Concentrated attack, the attacker needs to divide its attack budget, i.e. number of poisoning points, to points with $+1$ and -1 labels. We try three different splits – all $+1$ points, half-half, all -1 points – and report the best in terms of poisoning effect.

As mentioned in Section 6.5, Concentrated attack also needs to adapt to the online setting by imposing an order to set of poisoning examples generated for the offline setting. In positive first order, the attacker appends all the points with $+1$ label to the data stream before appending points with -1 labels. In negative first order, the attacker appends all the points with -1 label to the data stream first instead. In random order, the points are shuffled and appended to the data stream. We try 100 different random orders, and report the best among the positive-first, negative-first and random orders in terms of poisoning effect.

For poisoning the initialization set used by Slab defense, the attacker also needs to divide its attack budget to points with $+1$ and -1 labels. We divide the budget proportional to the fraction of $+1$ and -1 points in the clean initialization set.

C.2.3 Additional Experiment Results and Plots

In this section, we present additional experiment results and plots mentioned in the main body of paper.

The Semi-Online Setting

Error Rate of the Final Model on Clean Test Examples In Section 6.5, we present the cosine similarity between the final model θ and the target model θ^* against defense parameter τ. Since the target θ^* has high error rate on test examples, we also plot the test error rate of θ on clean examples in Figure C.1. The trend matches the cosine similarity reported in Section 6.5 – the final model has low error rate when in hard regime,

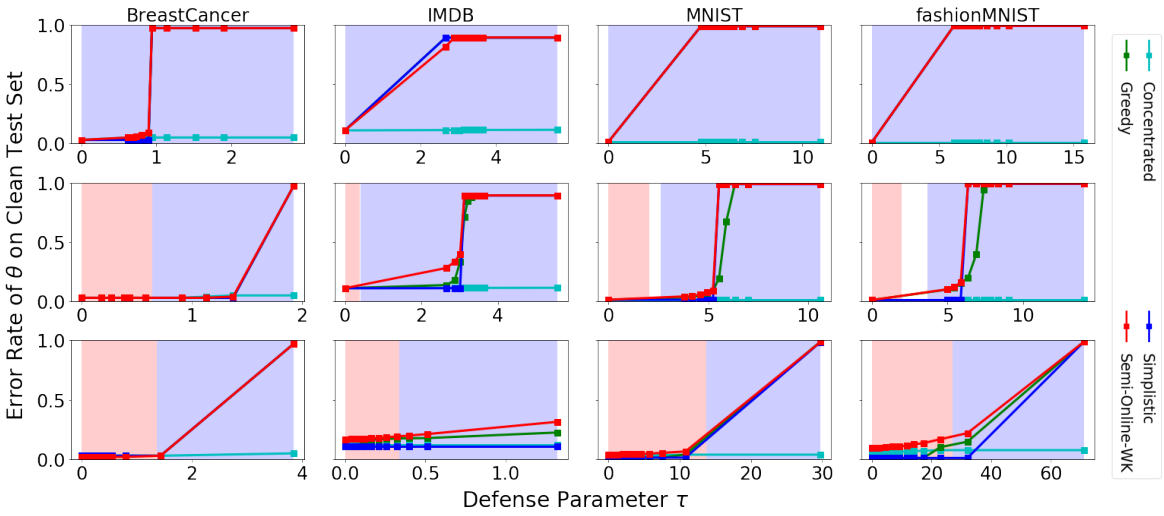

Figure. C.1: Plot of θ's error rate on clean test examples against defense parameter τ for the semi-online attacks. Small τ means stronger defense. Larger error rate means more successful attack. Red background region indicates hard regime, while blue background indicates easy regime. **Top to bottom**: L_2-norm, L_2-distance-to-centroid and Slab defense. **Left to Right**: BreastCancer, MNIST, fashionMNIST, IMDB.

high error rate for large τ in easy regime, and intermediate error rate for small τ in easy regime.

Cosine Similarity and Test Error Rate for Large Learning Rate. In Section 6.5, we claim that the attacker cannot make $\cos(\theta, \theta^*)$ close to 1 for small τ in easy regime because smaller τ increases the multiplicative factor C in the theory. To validate our claim, we use a larger learning rate $\eta = 1$ for all tasks, which reduces the factor C. Figure C.2 and C.3 show the cosine similarity to target model and the test error rate against defense parameter τ respectively. The result matches our expectation as for large η, a semi-online attacker can make θ close to θ^* even for small τ in easy regime. On the other hand, it still cannot make θ in the same direction as θ^* when in hard regime, and the test error rate remains low in hard regime.

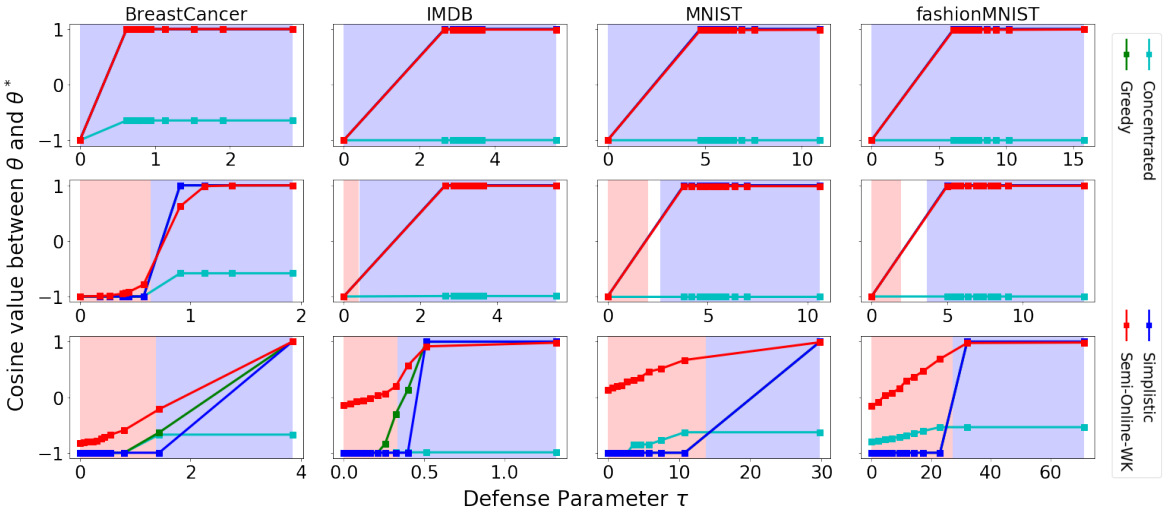

Figure. C.2: Plot of $\cos(\theta, \theta^*)$ against defense parameter τ for the semi-online attacks with larger learning rate ($\eta = 1$). Small τ means stronger defense. Larger $\cos(\theta, \theta^*)$ means more successful attack. Red background region indicates hard regime, while blue background indicates easy regime. **Top to bottom**: L_2-norm, L_2-distance-to-centroid and Slab defense. **Left to Right**: BreastCancer, MNIST, fashionMNIST, IMDB.

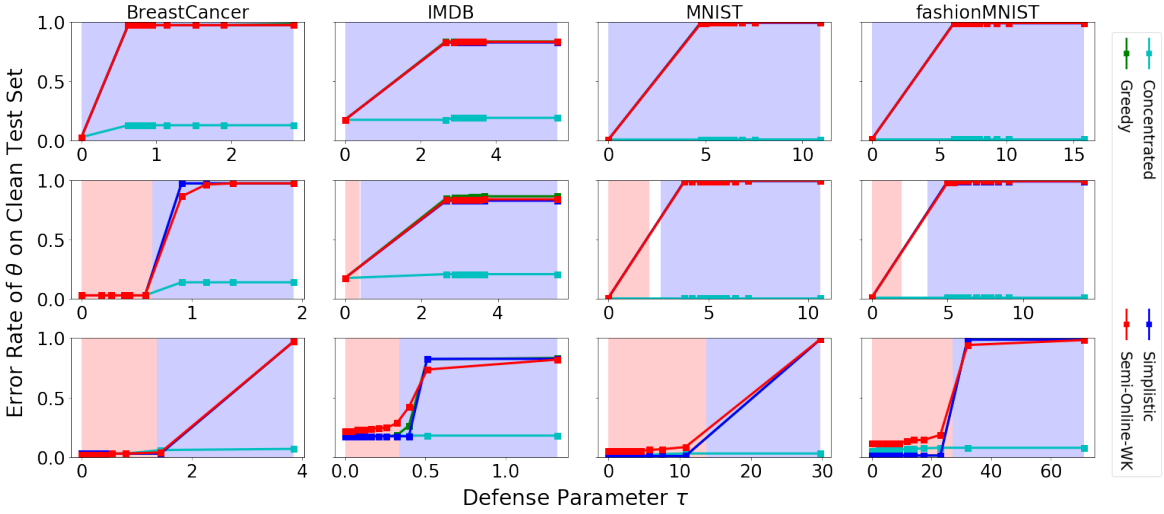

Figure. C.3: Plot of θ's error rate on clean test examples against defense parameter τ for the semi-online attacks with larger learning rate ($\eta = 1$). Small τ means stronger defense. Larger error rate means more successful attack. Red background region indicates hard regime, while blue background indicates easy regime. **Top to bottom**: L_2-norm, L_2-distance-to-centroid and Slab defense. **Left to Right**: BreastCancer, MNIST, fashionMNIST, IMDB.

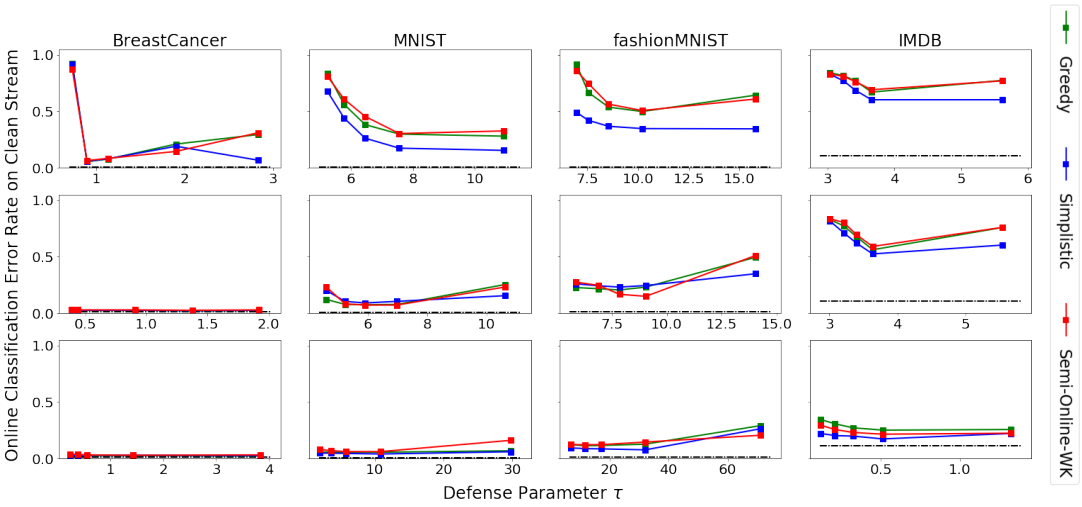

Figure. C.4: Online classification error rate against defense parameter τ when 20% of the examples in the data stream comes from the attackers. The dashdot line shows the error rate of the offline optimal classifier. Small τ means the defense filters out more examples, both clean and poisoned. Larger online classification error means more successful attack. **Top to bottom**: L_2-norm, L_2-distance-to-centroid and Slab defense. **Left to Right**: BreastCancer, MNIST, fashionMNIST, IMDB. The data sets from left to right are in decreasing order of difficulty for successful poisoning attacks.

The Fully-Online Setting

In Section 6.5, we consider a mild fully-online attacker which poisons 10% of the data stream. We also try a more powerful fully-online attack that can poison 20% of the data stream. Figure C.4 shows the online classification error rate in the presence of this more powerful attacker. The error rate is high for the weak L_2 norm defense. Slab is still able to keep the error rate low and is thus the strongest defense among all three. The L_2-distance-to-centroid defense's error rate is in between, and can typically keep the error rate below the fraction of poisoning examples in the stream. Meanwhile, the difficulty levels of poisoning attack over the four data sets are in the same order, with BreastCancer be the hardest and IMDB the easiest.

Bibliography

[ABE+16] Laurent Amsaleg, James Bailey, Sarah Erfani, Teddy Furon, Michael E Houle, Miloš Radovanović, and Nguyen Xuan Vinh. The vulnerability of learning to adversarial perturbation increases with intrinsic dimensionality. 2016.

[ADHHO18] Abdullah Al-Dujaili, Alex Huang, Erik Hemberg, and Una-May O'Reilly. Adversarial deep learning for robust detection of binary encoded malware. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 76–82. IEEE, 2018.

[ASH+14] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[AZB16] Scott Alfeld, Xiaojin Zhu, and Paul Barford. Data poisoning attacks against autoregressive models. In *AAAI*, pages 1452–1458, 2016.

[BCM+13] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Machine Learning and Knowledge Discovery in Databases*, pages 387–402. Springer, 2013.

[BL17] Cody Burkard and Brent Lagesse. Analysis of causative attacks against svms learning from data streams. In *Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics*, pages 31–36. ACM, 2017.

[BNL12] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *29th International Coference on International Conference on Machine Learning (ICML)*, Edinburgh, Scotland, June 2012.

[BPM17] Richard Baumann, Mykolai Protsenko, and Tilo Müller. Anti-proguard: Towards automated deobfuscation of android apps. In *Proceedings of the 4th Workshop on Security in Highly Connected IT Systems*, SHCIS '17, pages 7–12, New York, NY, USA, 2017. ACM.

[BPRB+13] Battista Biggio, Ignazio Pillai, Samuel Rota Bulò, Davide Ariu, Marcello Pelillo, and Fabio Roli. Is data clustering in adversarial settings secure? In

Proceedings of the 2013 ACM workshop on Artificial intelligence and security, pages 87–98. ACM, 2013.

[BR17] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *arXiv preprint arXiv:1712.03141*, 2017.

[BRTV16] Benjamin Bichsel, Veselin Raychev, Petar Tsankov, and Martin Vechev. Statistical deobfuscation of android applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 343–355, New York, NY, USA, 2016. ACM.

[CBL06] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

[CD10] Kamalika Chaudhuri and Sanjoy Dasgupta. Rates of convergence for the cluster tree. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 343–351. Curran Associates, Inc., 2010.

[CD14] Kamalika Chaudhuri and Sanjoy Dasgupta. Rates of convergence for nearest neighbor classification. In *Advances in Neural Information Processing Systems*, pages 3437–3445, 2014.

[CH67] T. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[CJK+07] Mihai Christodorescu, Somesh Jha, Johannes Kinder, Stefan Katzenbeisser, and Helmut Veith. Software transformations to improve malware detection. *Journal in Computer Virology*, 3:253–265, 10 2007.

[CLD11] Kevin Coogan, Gen Lu, and Saumya Debray. Deobfuscation of virtualization-obfuscated software: A semantics-based approach. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 275–284, New York, NY, USA, 2011. ACM.

[CSL+08] Gabriela F Cretu, Angelos Stavrou, Michael E Locasto, Salvatore J Stolfo, and Angelos D Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 81–95. IEEE, 2008.

[CW17] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, May 2017.

[CZS+17] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks

without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26. ACM, 2017.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DGKL94] Luc Devroye, Laszlo Gyorfi, Adam Krzyzak, and Gabor Lugosi. On the strong universal consistency of nearest neighbor regression function estimates. *The Annals of Statistics*, pages 1371–1385, 1994.

[DHPZ19] Sanjoy Dasgupta, Daniel Hsu, Stefanos Poulis, and Xiaojin Zhu. Teaching a black-box learner. In *International Conference on Machine Learning*, pages 1547–1555, 2019.

[DKK+17] Ilias Diakonikolas, Gautam Kamath, Daniel M Kane, Jerry Li, Ankur Moitra, and Alistair Stewart. Being robust (in high dimensions) can be practical. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 999–1008. JMLR. org, 2017.

[DKK+18a] Ilias Diakonikolas, Gautam Kamath, Daniel M Kane, Jerry Li, Ankur Moitra, and Alistair Stewart. Robustly learning a gaussian: Getting optimal error, efficiently. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2683–2702. Society for Industrial and Applied Mathematics, 2018.

[DKK+18b] Ilias Diakonikolas, Gautam Kamath, Daniel M Kane, Jerry Li, Jacob Steinhardt, and Alistair Stewart. Sever: A robust meta-algorithm for stochastic optimization. *arXiv preprint arXiv:1803.02815*, 2018.

[DKS16] Ilias Diakonikolas, Daniel M Kane, and Alistair Stewart. Efficient robust proper learning of log-concave distributions. *arXiv preprint arXiv:1606.03077*, 2016.

[DW77] Luc P Devroye and Terry J Wagner. The strong uniform consistency of nearest neighbor density estimates. *The Annals of Statistics*, pages 536–540, 1977.

[ERLD17] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751*, 2017.

[ETT+17] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. Exploring the landscape of spatial robustness. *arXiv preprint arXiv:1712.02779*, 2017.

[FHT00] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[GMR+19] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):93, 2019.

[GPM+16] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick D. McDaniel. Adversarial perturbations against deep neural networks for malware classification. *CoRR*, abs/1606.04435, 2016.

[GPM+17] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*, pages 62–79. Springer, 2017.

[GSS14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[HA17] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in Neural Information Processing Systems*, pages 2266–2276, 2017.

[Haz16] Elad Hazan. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[HP18] Hossein Hosseini and Radha Poovendran. Semantic adversarial examples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1614–1619, 2018.

[HPG+17] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[HPK01] Mark Handley, Vern Paxson, and Christian Kreibich. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*, SSYM'01, Berkeley, CA, USA, 2001. USENIX Association.

[HS16] Wenyi Huang and Jack W Stokes. Mtnet: a multi-task neural network for dynamic malware classification. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 399–418. Springer, 2016.

[HSW+19] Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. Achieving verified robustness to symbol substitutions via interval bound propagation. *arXiv preprint arXiv:1909.01492*, 2019.

[HT18] Weiwei Hu and Ying Tan. Black-box attacks against rnn based malware detection algorithms. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[ITAW18] Inigo Incer, Michael Theodorides, Sadia Afroz, and David Wagner. Adversarially robust malware detection using monotonic classification. In *the Fourth ACM International Workshop on Security and Privacy Analytics (IWSPA)*, Tempe, AZ, USA, Mar. 2018.

[JOB+18] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 19–35. IEEE, 2018.

[JSD+17] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC, Aug. 2017.

[KL17] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730*, 2017.

[KP95] S. Kulkarni and S. Posner. Rates of convergence of nearest neighbor estimation under arbitrary sampling. *IEEE Transactions on Information Theory*, 41(4):1028–1039, 1995.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KSL18] Pang Wei Koh, Jacob Steinhardt, and Percy Liang. Stronger data poisoning attacks break data sanitization defenses. *arXiv preprint arXiv:1811.00741*, 2018.

[KW15] Aryeh Kontorovich and Roi Weiss. A bayes consistent 1-nn classifier. In *Artificial Intelligence and Statistics Conference*, 2015.

[LAG+19] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672. IEEE, 2019.

[LDH+17] Weiyang Liu, Bo Dai, Ahmad Humayun, Charlene Tay, Chen Yu, Linda B Smith, James M Rehg, and Le Song. Iterative machine teaching. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2149–2158. JMLR. org, 2017.

[LDL+17] Weiyang Liu, Bo Dai, Xingguo Li, Zhen Liu, James M Rehg, and Le Song. Towards black-box iterative machine teaching. *arXiv preprint arXiv:1710.07742*, 2017.

[LHL+17] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. *arXiv preprint arXiv:1703.06748*, 2017.

[Lic13] M. Lichman. UCI machine learning repository, 2013.

[LLW91] Nicholas Littlestone, Philip M Long, and Manfred K Warmuth. On-line learning of linear functions. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 465–475. ACM, 1991.

[LWC+19] Qi Lei, Lingfei Wu, Pin-Yu Chen, Alexandros G Dimakis, Inderjit S Dhillon, and Michael Witbrock. Discrete adversarial attacks and submodular optimization with applications to text classification. *Systems and Machine Learning (SysML)*, 2019.

[LWSV16] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. In *Advances in neural information processing systems*, pages 1885–1893, 2016.

[MC17] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 135–147, 2017.

[MDFF15] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *arXiv preprint arXiv:1511.04599*, 2015.

[MDP+11] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[MGBD+17] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38. ACM, 2017.

[MMS+17] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *stat*, 1050:9, 2017.

[MMS+18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations (ICLR)*, Vancouver, Canada, Apr. 2018.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

[MZ15] Shike Mei and Xiaojin Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *AAAI*, pages 2871–2877, 2015.

[NBC+08] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D Joseph, Benjamin IP Rubinstein, Udam Saini, Charles A Sutton, J Doug Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. *LEET*, 8:1–9, 2008.

[NKS06] James Newsome, Brad Karp, and Dawn Song. Paragraph: Thwarting signature learning by training maliciously. In *International Workshop on Recent Advances in Intrusion Detection*, pages 81–105. Springer, 2006.

[NPXNR14] Andrew Newell, Rahul Potharaju, Luojie Xiang, and Cristina Nita-Rotaru. On the practicality of integrity attacks on document-level sentiment analysis. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pages 83–93. ACM, 2014.

[OPL13] J. Ouellette, A. Pfeffer, and A. Lakhotia. Countering malware evolution using cloud-based learning. In *8th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 85–94, Fajardo, Puerto Rico, USA, Oct 2013.

[PCG+17] Nicolas Papernot, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Fartash Faghri, Alexander Matyasko, Karen Hambardzumyan, Yi-Lin Juang, Alexey Kurakin, Ryan Sheatsley, Abhibhav Garg, and Yen-Chen Lin. cleverhans v2.0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 2017.

[PMG16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[PMG+17] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. In *Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security*, 2017.

[PMGGL18] Andrea Paudice, Luis Muñoz-González, Andras Gyorgy, and Emil C Lupu. Detection of adversarial training examples in poisoning attacks through anomaly detection. *arXiv preprint arXiv:1802.03041*, 2018.

[PMJ+16] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387, Saarbrucken, GERMANY, Mar. 2016.

[PSS+15] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas. Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brisbane, QLD, Australia, April 2015.

[PYZ18] Li Pengcheng, Jinfeng Yi, and Lijun Zhang. Query-efficient black-box attack by active learning. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 1200–1205. IEEE, 2018.

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[ŘS10] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[RSER19] Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach. Defense methods against adversarial examples for recurrent neural networks. *arXiv preprint arXiv:1901.09963*, 2019.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.

[RSRE18] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Generic black-box end-to-end attack against state of the art api call based malware classifiers. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 490–510. Springer, 2018.

[SB16] Aleieldin Salem and Sebastian Banescu. Metadata recovery from obfuscated programs using machine learning. In *Proceedings of the 6th Workshop on Software Security, Protection, and Reverse Engineering*, SSPREW '16, pages 1:1–1:11, New York, NY, USA, 2016. ACM.

[SHN+18] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*, pages 6103–6113, 2018.

[SKL17] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems*, pages 3520–3532, 2017.

[SMK+18] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. When does machine learning { FAIL } ? generalized transferability for evasion and poisoning attacks. In *27th { USENIX } Security Symposium ({USENIX} Security 18)*, pages 1299–1316, 2018.

[SS12] Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

[SSSS17] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 3–18. IEEE, 2017.

[Sto77] C. Stone. Consistent nonparametric regression. *Annals of Statistics*, 5:595–645, 1977.

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *CoRR*, abs/1703.01365, 2017.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[TZJ+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *USENIX Security Symposium*, pages 601–618, 2016.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[WC18] Yizhen Wang and Kamalika Chaudhuri. Data poisoning attacks against online learning. *arXiv preprint arXiv:1808.08994*, 2018.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 2017.

[YCH+18] Puyudi Yang, Jianbo Chen, Cho-Jui Hsieh, Jane-Ling Wang, and Michael I Jordan. Greedy attack and gumbel attack: Generating adversarial examples for discrete data. *arXiv preprint arXiv:1805.12316*, 2018.

[ZAGZ17] Mengchen Zhao, Bo An, Wei Gao, and Teng Zhang. Efficient label contamination attacks against black-box learning models. In *IJCAI*, pages 3945–3951, 2017.

[Zin03] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 928–936, 2003.

[ZSAL19] Wei Emma Zhang, Quan Z Sheng, A Alhazmi, and C Li. Adversarial attacks on deep learning models in natural language processing: A survey. *arXiv preprint arXiv:1901.06796*, 2019.

[ZYJ+19] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573*, 2019.

[ZZ19] Xuezhou Zhang and Xiaojin Zhu. Online data poisoning attacks. *arXiv preprint arXiv:1903.016662*, 2019.