Behavior of Machine Learning Algorithms in Adversarial Environments

Blaine Nelson

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2010-140
http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-140.html

November 23, 2010

Behavior of Machine Learning Algorithms in Adversarial Environments

by

Blaine Alan Nelson

A dissertation submitted in partial satisfaction

of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Anthony D. Joseph, Chair
Professor J. D. Tygar
Professor Peter L. Bartlett
Professor Terry Speed

Fall 2010

Behavior of Machine Learning Algorithms in Adversarial Environments

Blaine Alan Nelson

Abstract

Behavior of Machine Learning Algorithms in Adversarial Environments

by

Blaine Alan Nelson

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Anthony D. Joseph, Chair

Machine learning has become a prevalent tool in many computing applications and modern enterprise systems stand to greatly benefit from learning algorithms. However, one concern with learning algorithms is that they may introduce a security fault into the system. The key strengths of learning approaches are their adaptability and ability to infer patterns that can be used for predictions or decision making. However, these assets of learning can potentially be subverted by adversarial manipulation of the learner's environment, which exposes applications that use machine learning techniques to a new class of security vulnerabilities.

I analyze the behavior of learning systems in adversarial environments. My thesis is that learning algorithms are vulnerable to attacks that can transform the learner into a liability for the system they are intended to aid, but by critically analyzing potential security threats, the extent of these threat can be assessed, proper learning techniques can be selected to minimize the adversary's impact, and failures of system can be averted.

I present a systematic approach for identifying and analyzing threats against a machine learning system. I examine real-world learning systems, assess their vulnerabilities, demonstrate real-world attacks against their learning mechanism, and propose defenses that can successful mitigate the effectiveness of such attacks. In doing so, I provide machine learning practitioners with a systematic methodology for assessing a learner's vulnerability and developing defenses to strengthen their system against such threats. Additionally, I also examine and answer theoretical questions about the limits of adversarial contamination and classifier evasion.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost I would like to thank my advisor, Anthony Joseph, for the encouragement, guidance, and support he has offered me throughout my graduate career. Anthony taught me how to conduct successful research in a multi-disciplinary field and how to explore new fields of research.

I would like to thank Professor Doug Tygar for his guidance and insight throughout my graduate career. He made innumerable contributions to my development as a researcher and provided invaluable advice for pursuing research during and after my tenure at Berkeley.

I would like to thank Professor Peter Bartlett for providing constructive insights into my research and helping to guide the overall direction my research endeavors.

I would like to thank Professor Terry Speed for being on my qualifying exam and dissertation committees, and for his encouragement and feedback.

For encouraging me to pursue graduate research and encouraging my ambitions, I would like to thank Professor John Rose. For providing me with useful feedback, discussions, and insights, I would like to thank Professor Satish Rao, Professor Michael Jordan, Professor Laurent El Ghaoui, Professor Gert Lanckriet, and Professor Charles Sutton.

Many others have helped me over the course of my graduate career. I cannot thank all these individuals enough for their support, but I would like to call attention to a few who were most instrumental to this undertaking. I would particularly like to thank Marco Barreno and Ben Rubenstein for their ideas, hard work, and dedication that made this dissertation possible. Marco and Ben made critical contributions to my dissertation projects and were extraordinary collaborators and friends. I would also like to thank Russell Sears, Peter Bodík, Arel Cordero, Alexandre Bouchard, Fabian Wauthier, Kurt Miller, Anil Sewani, Steve Martin, Ira Cohen, Marius Kloft, and Guillaume Obozinski for their feedback, discussions, and input that made this dissertation possible. For their collaboration on many research projects and persistent hard work, I thank Udam Saini, Kai Xai, Shing-hon Lau, Jack Chi, Anthony Tran, and Chris Cai.

Finally, I wish to thank my parents, Lonnie and Tricia Nelson, and my brother, Bryce Nelson, for providing unwavering support, advise on life, and assistance when I needed it. I would also like to thank Elizabeth Segran and Carolina Galleguillos for being good friends who always were willing to listen and commiserate with me. Without all of them, this work would not have been possible. Additionally, I wish to thank Mary Jane Sullivan, Arlen and June Maxfeldt, Bob and Kelly Balzer, Bob and Jane Sullivan, Peter and Mary Sullivan, and the rest of my extended family for reinforcing my pursuit of higher education.

Portions of this dissertation have appeared in previously published works [Barreno et al., 2006, 2010, Nelson et al., 2008, 2009, 2010a, Rubinstein et al., 2009a].

I gratefully acknowledge the support of my sponsors. This work was supported in part by TRUST (Team for Research in Ubiquitous Secure Technology), which receives support from the National Science Foundation (NSF award number CCF-0424422) and the following organizations: AFOSR (#FA9550-06-1-0244), BT, Cisco, DoCoMo USA Labs, EADS, ESCHER, HP,IBM, iCAST, Intel, Microsoft, ORNL, Pirelli, Qualcomm, Sun, Symantec,TCS, Telecom Italia, and United Technologies; in part by RAD Lab (Reliable Adaptive Distributed Systems Laboratory), which receives support from California state Microelectronics Innovation

and Computer Research Opportunities grants (MICRO ID#06-148 and #07-012) and the following organizations: Amazon Web Services, CISCO, Cloudera, eBay, Facebook, Fujitsu Labs of America, Google, Hewlett Packard, Intel, Microsoft, NetApp, SAP, Sun, VMWare, and Yahoo!; and in part by the cyber-DEfense Technology Experimental Research laboratory (DETERlab), which receives support from the Department of Homeland Security Homeland Security Advanced Research Projects Agency (HSARPA award #022412) and AFOSR (#FA9550-07-1-0501). The opinions expressed here are solely those of the author and do not necessarily reflect the opinions of any funding agency, the State of California, or the U.S. government.

Chapter 1

Introduction

Machine learning has become a prevalent tool in many computing applications. While learning techniques are already common for tasks such as natural language processing [*cf.*, Jurafsky and Martin, 2008], face detection [*cf.*, Zhao et al., 2003], and handwriting recognition [*cf.*, Plamondon and Srihari, 2000], they also have potentially far-reaching utility for many applications in security, networking, and large-scale systems as a vital tool for data analysis and autonomic decision making. As suggested by Mitchell [2006], learning approaches are particularly well-suited to domains where either the application *i*) is too complex to be designed manually or *ii*) needs to dynamically evolve. Many of the challenges faced in modern enterprise systems meet these criteria and stand to benefit from agile learning algorithms able to infer hidden patterns in large complicated datasets, adapt to new behaviors, and provide statistical soundness to decision-making processes. Indeed, learning components have been proposed for tasks such as performance modeling [*e.g.*, Bodík et al., 2010, 2009, Xu et al., 2004], enterprise-level network fault diagnosis [*e.g.*, Bahl et al., 2007, Cheng et al., 2007, Kandula et al., 2008], and spam detection [*e.g.*, Meyer and Whateley, 2004, Segal et al., 2004] but generally adoption is not yet widespread.

One potential concern with learning algorithms is that they may introduce a security fault into the system. The key strengths of learning approaches are their adaptability and ability to infer patterns that can be used for predictions or decision making. However, these assets of learning can potentially be subverted by adversarial manipulation of the learner's environment, which exposes applications that use machine learning techniques to a new class of security vulnerabilities; *i.e.*, learners are susceptible to a novel class of attacks that can cause the learner to disrupt the system it was intended to improve. Here I analyze the behavior of learning systems under duress in *security-sensitive domains*. My thesis is that learning algorithms are vulnerable to a myriad of attacks that can transform the learner into a liability for the system they are intended to aid, but by critically analyzing potential security threats, the extent of these threats can be assessed, proper learning techniques can be selected to minimize the adversary's impact, and failures of system can be averted.

In this dissertation, I investigate both the practical and theoretical aspects of applying machine learning to security domains and here I summarize the four components of my dissertation project: a taxonomy for qualifying the security vulnerabilities of a learner, two novel practical attack and defense scenarios, and a generalization of a paradigm for evading detection of a classifier. I present a framework for identifying and analyzing threats to

learners and use it to systematically explore the vulnerabilities of two learning systems. For these systems, I identify real-world threats, analyze the potential impact of each, and study learning techniques that significantly diminish their vulnerabilities. In doing so, I provide practitioners with guidelines to identify potential vulnerabilities and demonstrate improved learning techniques resilient to attacks. My research focuses on learning tasks in virus, spam, and network anomaly detection, but also is broadly applicable across many systems and security domains and has far-reaching implications for any system that incorporates learning. In the remainder of this chapter, I further motivate the need for a security analysis of machine learning algorithms and provide a brief history of the work that led me to this research and the lessons learned from it.

1.1 Motivation and Methodology

Machine learning techniques are being applied to a growing number of systems and networking problems. Of particular interest to my research work is the problem of detecting various types of anomalous system behavior; I refer to this area broadly as *malfeasance detection* and it includes spam, fraud, intrusion, and virus detection. For such a problem domain, machine learning techniques provide the ability for the system to respond more readily to evolving real-world data, both hostile and benign, and learn to identify or even possibly prevent undesirable behavior. As an example, network intrusion detection systems (NIDS) monitor network traffic to detect abnormal activities such as attempts to infiltrate or hijack hosts on the network. The traditional approach to designing a NIDS relied on an expert codifying rules defining normal behavior and intrusions [*e.g.* Paxson, 1999]. Because this approach often fails to detect novel intrusions, a variety of researchers have proposed incorporating machine learning techniques into intrusion detection systems [*e.g.*, Mahoney and Chan, 2002, Lazarevic et al., 2003, Mukkamala et al., 2002, Eskin et al., 2002]. Machine learning techniques offer the benefit that they can detect novel differences in traffic (which presumably represent attack traffic) by being trained on examples of innocuous (known good) and malicious (known bad) traffic. Learning approaches to malfeasance detection have also played a prominent role in modern spam filtering [*e.g.*, Meyer and Whateley, 2004, Segal et al., 2004] and also have been proposed as elements in virus and worm detectors [*e.g.*, Newsome et al., 2005, Stolfo et al., 2003, 2004], host-based intrusion detection systems (HIDS) [*e.g.*, Forrest et al., 1996, Hofmeyr et al., 1998, Mutz et al., 2006, Somayaji and Forrest, 2000, Warrender et al., 1999], and other types of fraud detection [*cf.*, Bolton and Hand, 2002].

However, using machine learning techniques introduces the possibility of an adversary, who maliciously exploits the unique vulnerabilities of a learning system. With growing financial incentives of cybercrime inviting ever more sophisticated adversaries, attacks against learners present a lucrative new means to disrupt the operations of or otherwise damage enterprise systems. This makes assessing the vulnerability of learning systems an essential problem to address in order to make learning methods effective and trustworthy in security-sensitive domains. An intelligent adversary can alter his approach based on knowledge of the learner's shortcomings or mislead it by cleverly crafting data to corrupt or deceive the learning process; *e.g.*, spammers have regularly adapted their messages to thwart or evade spam detectors. In this way, malicious users can subvert the learning process to disrupt a service or perhaps even compromise an entire system.

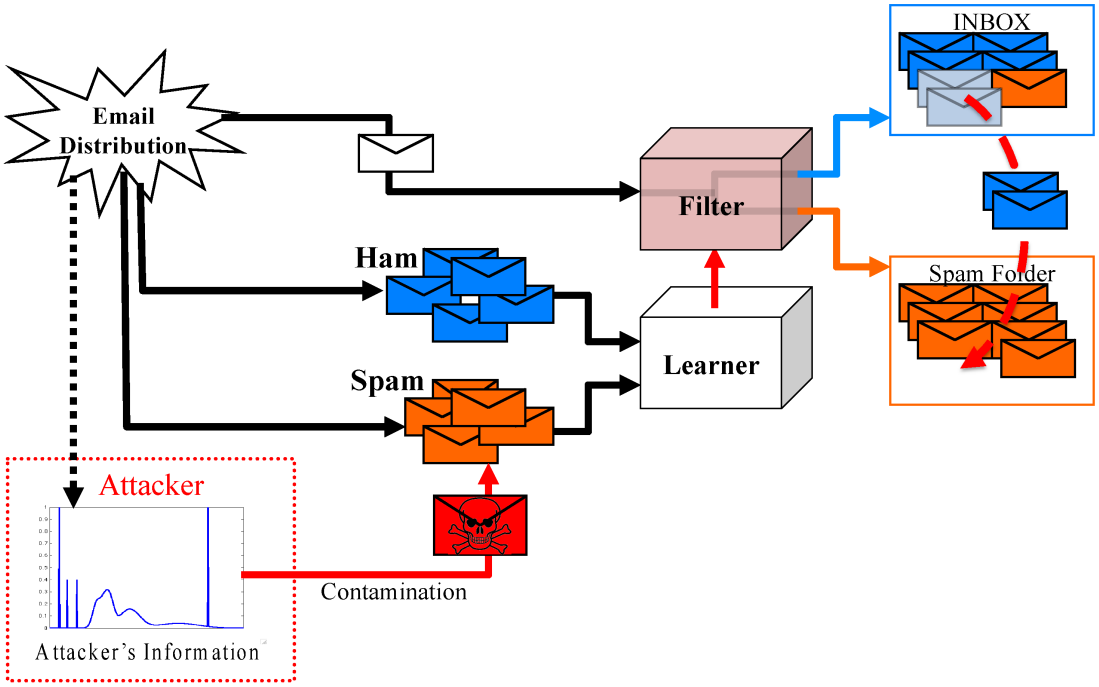
The primary flaw in learners that attackers can exploit lies in the assumptions made about the learner's data. Many common learning algorithms are predicated on the assumption that their training and evaluation data comes from a natural or well-behaved distribution that remains stationary over time, or at worst, changes slowly in a benign way (gradual drift). However, these assumptions are perilous in a security-sensitive domain—an application domain where a patient adversary has a motive and the capability to alter the data used by the learner for training or prediction. In such a domain, learners can be manipulated by an intelligent adversary capable of cleverly violating the learner's assumptions for their own gains making learning and adaptability into potential liabilities for the system rather than benefits. I analyze how learners behave in these settings and alternative methods that can bolster the system's resilience to an adversary.

I consider several potential dangers posed to a learning system. The primary threat is that an attacker can exploit the adaptive nature of a machine learning system to mis-train it and cause it to fail. Here, failure consists of causing the learning system to produce classification errors: if it misidentifies a hostile instance as benign, then the hostile instance is erroneously permitted through the security barrier; if it misidentifies a benign instance as hostile, then a permissible instance is erroneously rejected and normal user activity is interrupted. The adversarial opponent has the ability to design training data that will cause the learning system to produce rules that misidentify instances. If the system's performance sufficiently degrades, users will lose confidence in the system and abandon it or its failures may significantly compromise the integrity of the system. This threat raises several questions: *What techniques can a patient adversary use to mis-train or evade a learning system?* and *How can system designers assess the vulnerability of their system to vigilantly incorporate trustworthy learning methods?* I provide a framework for a system designer to thoroughly assess these threats and demonstrate how it can be applied to evaluate real-world systems.

Developing robust learning and decision making processes is of interest in its own right, but for security practitioners, it is especially important. To effectively apply machine learning as a general tool for reliable decision-making in computer systems, it is necessary to investigate how these learning techniques perform when exposed to adversarial conditions. Without an in-depth understanding of the performance of these algorithms in an adversarial setting, the systems will not be trusted and will fail to garner wider adoption. Worse yet, a vulnerable system could be exploited and disaffect practitioners from using learning systems in the future. When a learning algorithm performs well under a realistic adversarial setting, it is an algorithm for *secure learning*. Of course, whether an algorithm's performance is acceptable is a highly subjective judgement that depends both on the constraints placed on the adversary and on the job the algorithm is tasked with performing. This raises two fundamental questions: *What are the relevant security criteria to evaluate the security of a learner in a particular adversarial environment?* and *Are there machine learning techniques capable of satisfying the security requirements of a given problem domain and how can such a learner be designed or selected?* I demonstrate how learning systems can be systematically assessed and how learning techniques can be selected to diminish the potential impact of an adversary.

I now present three high-level examples that describe different attacks against a learning system. Each of these are later comprehensively analyzed in Chapters 4, 5, and 6, but here I summarize the setting of each to lay a foundation for the reader. In each synopsis I motivate the learning task and the goal of the adversary. I then briefly describe plausible attacks that align with these goals.

Example 1.1 (Spam Filter and Data Sanitization)

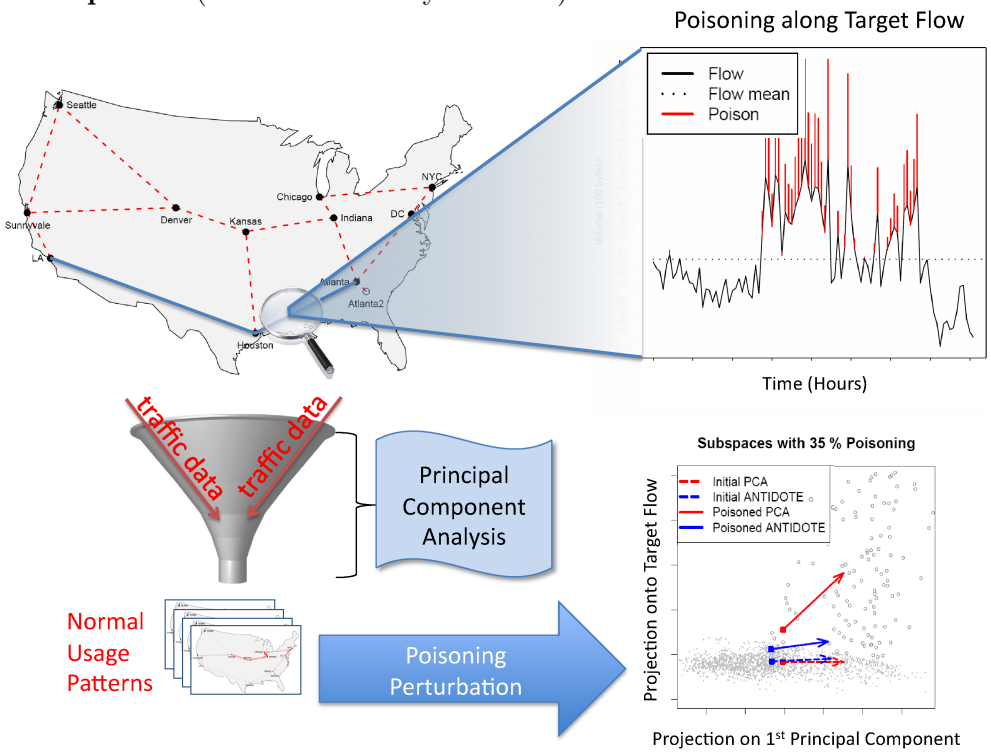


Spam filtering is one of the most common applications of machine learning. In this problem, a set of known good email (ham) and unwanted email (spam) are used to train a spam filter. The learning algorithm identifies relevant characteristics that distinguish spam from ham (*e.g.*, tokens such as "Viagra", "Cialis", and "Rolex" or envelope-based features) and constructs a classifier that combines observed evidence of spam to make a decision about whether a newly received message is a spam or ham.

Spam filters have proven to be successful at correctly identifying and removing spam messages from a user's regular messages. This has inspired spammers to regularly attempt to evade detection by *obfuscating* their spam messages to confuse common filters. However, spammers can also corrupt the learning mechanism. As pictured in the diagram above, a clever spammer can use information about the email distribution to construct clever *attack spam* messages that, when trained on, will cause the spam filter to misclassify the user's desired messages as spam. Ultimately, the spammers goal here is to cause the filter to become so unreliable that the user can no longer trust that his filter has accurately classified the messages and must sort through spam to ensure that important messages are not erroneously filtered.

In Chapter 4, I explore several variants of this attack based on different goals for the spammer and different amounts of information available to him. This attack proves to be quite effective: if a relatively small number of attack spam are trained on, the accuracy of the filter is significantly reduced. However, I also show that a simple data sanitization technique that was designed to detect deleterious messages is effective in preventing many of these attacks. In this case, the attacker's success depends primarily on the scope of their goal to disrupt the user's email.

Example 1.2 (Network Anomaly Detector)

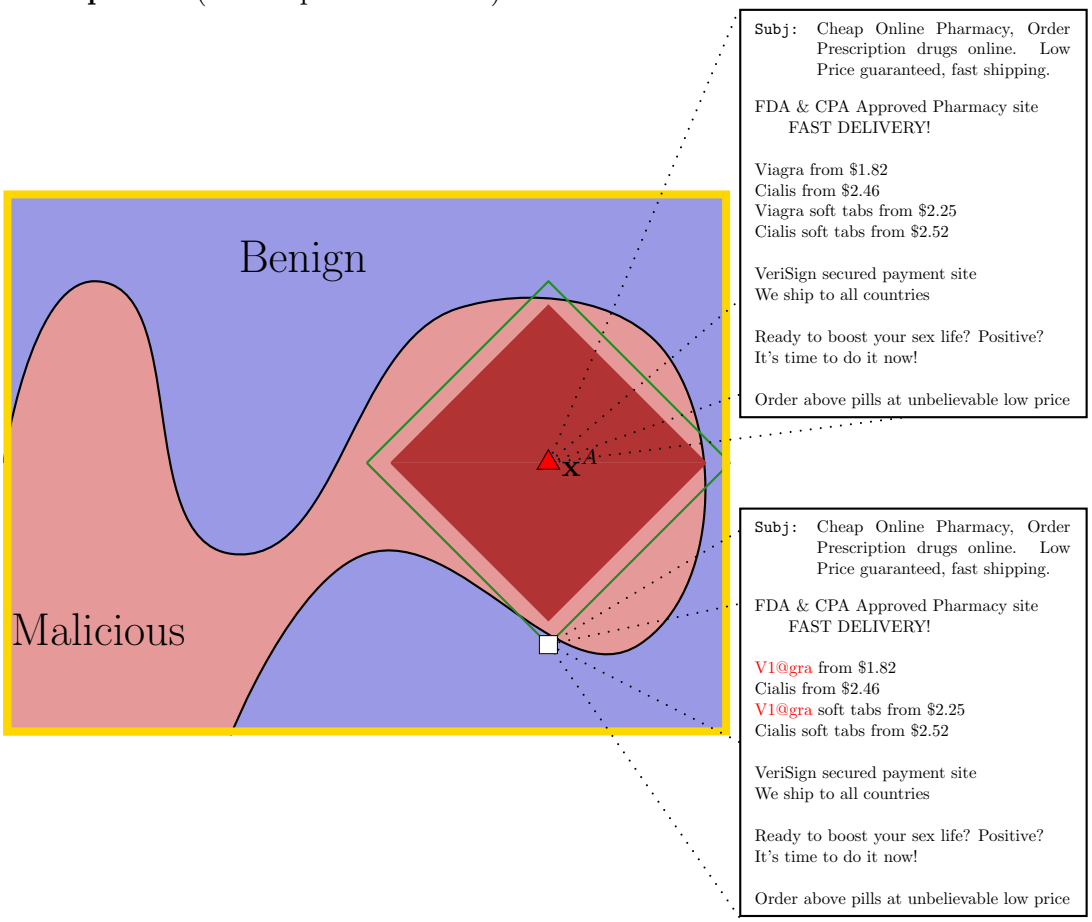


Machine learning techniques have also been proposed by Lakhina et al. [2004b] for detecting network volume anomalies such as *denial-of-service (DoS) attacks*. Their proposal uses a learning technique known as principal component analysis (PCA) to estimate normal traffic patterns and identify anomalous activity in the network. However, as with the spam filter in the previous example, this technique is also susceptible to contamination.

As depicted in the above diagram, PCA extracts patterns from traffic observed flowing over a backbone communications network to construct a normal model of it. This model is subsequently used to detect DoS attacks. Thus, an adversary determined to launch a DoS attack must first evade this detector. A crafty adversary can successfully evade detection by mis-training the detector. He can systematically inject chaff traffic into the network that is designed to make his target flow align with the normal model—this chaff (depicted in red in the top-right figure) is added along the target flow to increase variance. The resulting perturbed model (see the bottom-right figure) is unable to detect DoS attacks along the target flow.

I explore attacks against the PCA-base detector in Chapter 5 again based on different sources of information available to the adversary. Attacks against PCA prove to be effective—they successfully increase its rate of mis-detection eight to ten-fold. I also explore an alternative detection approach called ANTIDOTE designed to be more resilient to chaff. The evasion success rate for the same attacks against ANTIDOTE is roughly halved compared to the PCA-based approach. However, resilience to poisoning comes at a price—ANTIDOTE is less effective on non-poisoned data than the original detector.

Example 1.3 (Near-Optimal Evasion)



In addition to misleading learning algorithms, attackers also have an interest in evading detectors by making their miscreant activity undetectable. As previously mentioned in Example 1.1, this practice is already common in the spam filtering domain where spammers attempt to evade the filter by *i*) obfuscating words indicative of spam to human-recognizable misspellings; *e.g.*, "Viagra" to"V1@gra" or "Cialis" to "Gia|is", *ii*) using clever HTML to make the content difficult to parse, *iii*) adding words or text from other sources unrelated to the spam, and *iv*) embedding images that contains the spam message. All of these techniques can be used to evade spam filters, but they also are costly for the spammer—altering his spam can make the message less profitable as the distortions reduce the message's legibility or its accessibility. Thus, in evading the filter, the spammer would like to minimally modify their messages, but for a dynamically learned filter, the spammer does not know the learned filtering rules. Instead, the spammer constructs test spams that he uses to probe the filter and refine his modifications according to some cost on them. This raises the following question: *How difficult is it for the spammer to optimally evade the filter by querying it?*

The *near-optimal evasion problem*, which I examine in Chapter 6, formalizes this question in terms of the query complexity required by the spammer to evade a particular family of classifiers. I study the family of *convex-inducing classifiers* and I show that there are efficient algorithms for near-optimal evasion under certain ℓ_p cost functions.

1.2 Guidelines from Computer Security

To assess the vulnerabilities of learning systems, I built on many principles established in traditional computer security. The area of computer security is a broad field with many facets and only a subset of them are pertinent to my work. In great generality, computer security is concerned with quantifying, managing, and reducing the risks associated with computer systems and their usage. Traditional topics in security include cryptography, authentication, secure channels, covert channels, defensive programming practices, static code analysis, network security, and operating system security and traditional (code-based) vulnerabilities include buffer overflows, format string vulnerabilities, cross application scripting, code injection attacks, and privilege escalation. Unlike classical security settings, attacks against a learning system exploit the adaptive nature of the learning system. Not only can the adversary exploit existing flaws in the learner, he can also mislead the learner to create new vulnerabilities. Nonetheless, classical security principles are also applicable for analyzing machine learning algorithms. Particularly, the principles of *proactively studying attacks*, *Kerckhoffs' Principle*, *conservative design*, and *formal threat modeling* are the foundation of my approach.

Proactive Analysis: The first guideline from computer security is to conduct proactive studies to anticipate potential attacks before a system is deployed or widely used. Analysis of and open debate about the security of a system provide a level of confidence in it and identifying vulnerabilities before deployment can prevent costly patches, rewrites, or recalls of flawed systems. My dissertation is a proactive study in the sense that I am exploring the vulnerabilities of learning systems to identify threats before major systems are damaged or compromised and, in exposing these vulnerabilities, I also offer alternative systems that thwart or mitigate them. Further, I provide general guidelines to system designers to aid them in analyzing the vulnerabilities of a proposed learning system so that learning can be deployed as an effective and reliable component even in critical systems.

Kerckhoffs' Principle: The second guideline often referred to as *Kerckhoffs' Principle* [Kerckhoffs, 1883] is that the security of a system should not rely on unrealistic expectations of secrecy. Depending on secrets to provide security is a dangerous policy because if these secrets are exposed the security of the system is immediately compromised. Ideally, secure systems should make minimal assumptions about what can realistically be kept secret from a potential attacker. The field of cryptography has embraced this general principle by demanding open algorithms that only require a secret *key* to provide security or privacy. I apply this principle to analyzing machine learning systems throughout this dissertation primarily by assuming that the adversary is aware of the learning algorithm and can obtain some degree of information about the data used to train the learner. However, determining the appropriate degree of secrecy that is feasible for secure machine learning systems is a difficult question, which I discuss further in Chapter 7. In each of the chapters of this thesis, I consider various levels of information that the adversary potentially obtains, and I assess how the adversary can best utilize this information to achieve their objective against the learner. In doing so, I demonstrate the impact of different levels of threat and show the value an adversary obtains from a particular source of information.

Conservative Design: The third principle I employ is that security analysis of a system should generally avoid placing unnecessary or unreasonable limitations on the adversary. All too often, major security compromises occur because designers failed to anticipate how powerful an adversary is or how well informed the adversary is. By assuming the adversary has the broadest possible powers, one can understand the worst-case threat posed by an adversary and users are less likely to be surprised by an attack by some unanticipated adversary. Conversely, though, analyzing the capabilities of an omnipotent limitless adversary reveals little about a learning system's behavior against realistic attackers and may lead to an unnecessarily bleak outlook on the feasibility of using learning at all. Instead, my approach is to construct an appropriate threat model to quantify the relationship between the adversary's effort and their effect on the system under a variety of different levels of threat including a worst-case adversary.

Threat Modeling: Finally, to analyze the vulnerabilities of machine learning systems, I follow the typical security practice of constructing a formal (attacker-centric) threat model. In most interesting settings, a completely secure system is infeasible and I do not attempt to achieve complete security in my work. Instead, my approach quantifies the *degree of security*—the level of security expected against an adversary with a certain set of objectives, capabilities, and incentives based on a *threat model*. Building a threat model allows the analyst to quantify the security of his system and design approaches to making the system reasonably secure.

To construct a threat model for a particular learning system, first the analyst quantifies the security setting and objectives of that system in order to develop criteria to measure success and quantify the level of security offered. Formalizing the risks and objectives allows the analyst to identify potential limitations of his system and potential attacks and focuses the analysis on immediate threats so as to avoid wasting effort protecting against nonexistent or ancillary threats. Next the analyst identifies potential adversarial goals, resources, and limitations. By examining the nature of anticipated adversaries and their goals, the analyst can quantify the effort required by the adversary to achieve their objectives. Based on this threat model, the analyst can finally analyze the security of his system and construct appropriate defenses against realistic forms of attack. Formal analysis provides a rigorous approach to security. Additionally, by formalizing the threats and security of a system, other analysts can critique the analyst's assumptions and suggest potential flaws in his design. This open process tends to improve a system's security.

In this dissertation, I analyze three separate security problems for machine learning systems. In each, I first specify the threat model posed and subsequently analyze the threat's impact and, where appropriate, I propose defenses against the threat. It is well-established in computer security that evaluating a system involves a continual process of first, determining classes of attacks on the system; second, evaluating the resilience of the system against those attacks; and third, strengthening the system against those classes of attacks. Throughout this dissertation, I follow exactly this model in evaluating the vulnerabilities of learning algorithms.

1.3 Historical Roadmap

Here I briefly summarize a series of projects that led me to study the adversarial machine learning setting and the lessons I learned in this early work that molded my approach to the topic. Prior to my dissertation project, I sought to use machine learning algorithms in various novel application domains that had adversarial elements. The first of these was a research project conducted at Duke University to detect anti-personnel landmines by identifying their unique electromagnetic signatures. I explored an approach based on neural networks trained to identify these devices based on readings from a metal detector. However, at the time, I did not consider the adversarial nature of landmine design or its potential impact on my detector. At Berkeley, I first explored applications of learning algorithms to computer systems and pursued a learning approach for detecting computer viruses, which was designed to capture requisite characteristics of viral behavior, but the inherently adversarial and adaptive nature of computer viruses led me to question our detector's longevity and security. I began scrutinizing this subject with colleagues with backgrounds in security, machine learning, and systems. This led us to design some of the elements of the detector to be robust against changing viral behaviors, to construct a theoretical model for analyzing the effect of contamination on hypersphere classifiers, and ultimately led to my doctoral project described in this thesis. Here, I briefly summarize the projects that proceeded my dissertation thus providing a chronology of the progression of my investigation into the security of learning algorithms.

1.3.1 Landmine Detection System

My first foray into applied machine learning was a project that explored neural network detectors for landmine detection and identification. This project extended research in existing signal analysis algorithms for landmine identification by examining a specific class of objects called anti-personnel devices that contain only a small amount of asymmetrically arranged metal causing their characteristic wide-band frequency responses to deviate significantly with small changes in relative position between a sensor and the landmine. This project explored methods to improve and extend the capabilities of existing algorithms by quantifying the limitations of electromagnetic induction (EMI) sensors for such objects and attempting to account for these deviations to properly identify anti-personnel devices when the sensor is not precisely centered over a device.

For this purpose, I used a set of neural network classifiers to learn the EMI responses characteristics that were unique for each type of anti-personnel device [Nelson et al., 2003]. The results of this effort met with limited success—while the neural net approach was effective it was outperformed by other signal processing techniques in several circumstances. Nonetheless, this project was my first attempt to use a learning algorithm in a security-sensitive domain. In this case, landmine makers played the role of designing anti-personnel devices to be difficult to detect using EMI sensors and undoubtedly, if these sensors coupled with techniques from learning theory or signal processing were able to effectively detect these landmines, the designers would further refine their designs to thwart these detectors as well. Not realizing the adversarial nature of this problem at the time, I reasoned about the neural network learner's effectiveness by measuring their detection capability on known landmine signatures without considering the potential for re-designs to evade detection—a

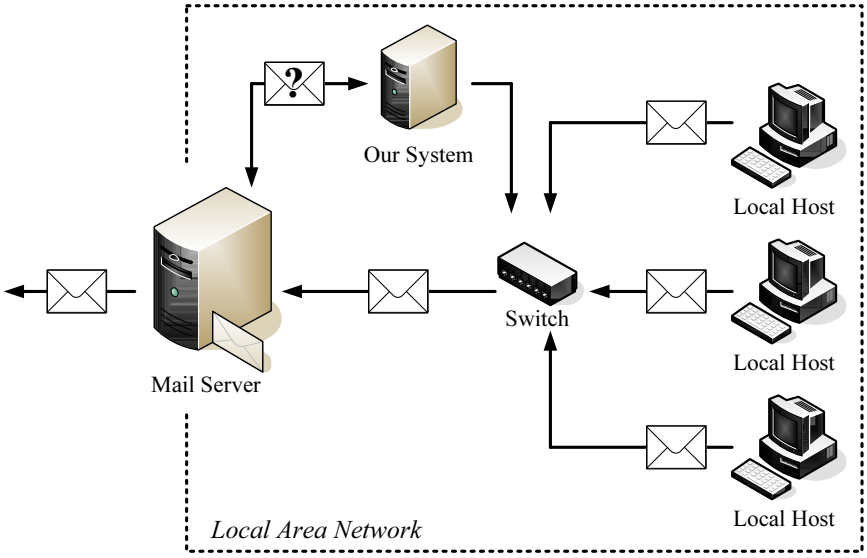
mistake often made by machine learning practitioners working in adversarial environments. Throughout this dissertation, I critique such oversights and both provide examples of how adversaries can effectively thwart learning systems and how learning systems can be more resilient to adversaries.

1.3.2 Virus Detection System

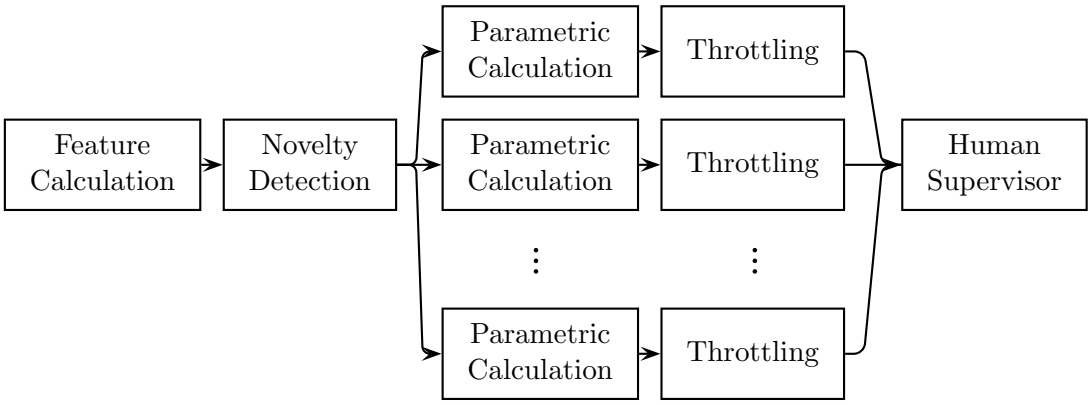
In my second learning-based application, I designed and implemented a dynamic virus detection system in collaboration with Karl Chen, Steve Martin, Anil Sewani, and Anthony Joseph [Martin, 2005, Sewani, 2005, Nelson, 2005]. In designing this system, my collaborators and I sought to counter the proliferation of novel email-based viruses and protect against obfuscation through polymorphisms. We demonstrated that this system could effectively detect a wide-variety of novel email-based viruses, because, unlike a rule-based signature detector, our system's learning component was able to quickly adapt to new threats. Here, I briefly give an overview of that virus detection system and the design considerations meant to remedy the fast spread of email viruses seen at that time. However, in designing and evaluating our system, I realized that learning systems could themselves become a significant vulnerability in a hostile environment—these considerations led to the systematic evaluation of the security of machine learning systems that I present throughout the rest of this work. Below, I briefly discuss the relevant details of this virus detection system and then critique its design from a security perspective to further motivate the security analysis described in the remainder of this dissertation.

The virus detection we designed was intended to counter the rapid proliferation of novel mass-mailing viruses that had made traditional signature-generation-based approaches untenable. The crux of this problem was the slow dissemination of the virus signature updates required by traditional systems to effectively halt viral spread. Such signatures were traditionally manually generated after samples of the novel virus were submitted to the anti-virus company for analysis—a process that left vulnerable systems unprotected to attack for hours or days whilst the virus spread. These response times were woefully inadequate to prevent devastating viral epidemics that wasted or damaged valuable network and computing resources by rapidly propagating as quickly as email could be sent.

Our detection strategy was a reactive approach; rather than detecting the incoming viral messages, we attempted to detect infected machines disseminating mass-emails—*i.e.*, an extrusion detection architecture as depicted in Figure 1.1(a). We chose an extrusion detection approach because even an effective intrusion-based virus detection system can fail (*e.g.*, detection can be circumvented if an externally infected machine is inadvertently brought behind the network's defenses) and expose the network to damages wrought by the virus from within. Further, we believed that the behavior of an infected machine was more detectable than the inbound infection because, once an infection succeeds, the compromised host tends to dramatically deviate from normal user behavior as the virus attempts to quickly propagate. Our system was thus designed to mitigate the effect of an infection once it occurs. By applying our approach at the network level, we hoped that quarantining based on the behavior of an infected machine would reduce the damage to mail-servers caused by an overwhelming stream of viral emails and isolate the infected hosts until they could be disinfected. Ultimately, we sought to thwart or mitigate the rapid proliferation strategy of email-based viruses.



(a) Virus Extrusion Detector Architecture



(b) Pipeline of Detectors for Virus Detection

Figure 1.1: Diagrams of the virus detection system architecture described in Martin [2005], Sewani [2005], Nelson [2005]. **(a)** The system was designed as an extrusion detector. Messages sent from local hosts are routed to our detector by the mail server for analysis—benign messages are subsequently sent whereas those identified as viral are quarantined for review by an administrator. **(b)** Within the detector, messages pass through a classification pipeline. After the message is vectorized, it is first analyzed by a one-class SVM novelty detector. Messages flagged as 'suspicious' are then re-classified by a per-user naive Bayes classifier. Finally, if the message is labeled as 'viral' a throttling module is used to determine when a host should be quarantined.

Our network solution to detecting viral activity in out-going email traffic used statistical learning techniques to monitor for sufficient deviations from normal email behavior using the architecture depicted in Figure 1.1(b). This design incorporated a set of features that represented the current state of email behavior. Using feature selection techniques, we chose a robust set of features that accurately distinguished normal behavior from viral behavior. A novelty detection algorithm was then used as a filter to isolate the majority of normal messages and a classification layer used past viral behavior to reduce false positives caused by traditional novelty detection alone. The resulting system was capable of quarantining hosts believed to be exhibiting viral behavior.

To detect mass-mailing viruses, we considered features that would best distinguish the infected and normal email behavior based on the following observations of email viruses: they must propagate the infection, they attempt to avoid detection, they have some degree of repetition between emails, and they have traditionally sent email at extraordinarily fast rates to propagate quickly. To capture these behaviors, we constructed two general types of features: *per-message features* that describe characteristics of a single message and *window-based features* that describe the behavior of the latest set of messages. These features are listed in the table below.

	Per-message Features	Window-based Features
1.	Whether or not the message is a reply or forward	Frequency of emails sent in the window
2.	Presence of HTML in the message	Number of unique email recipients
3.	Presence of HTML script tags or attributes in the message	Number of unique sender addresses
4.	Presence of embedded images in the message	Average number of words in the subject lines
5.	Presence of hyperlinks in the message	Average number of words in the bodies
6.	MIME types of file attachments in the message	Average number of characters in the subject lines
7.	Presence of binary attachments	Average number of characters in the bodies
8.	Presence of text attachments	Average word length in the messages
9.	The UNIX "magic number" of file attachments	Variance in number of words in the subject lines
10.	Total size of the message including attachments	Variance in number of words in the bodies
11.	Total size of files attached to the email	Variance in number of characters in the subject lines
12.	Number of files attached to the email	Variance in number of characters in the bodies
13.	Number of words in the message's subject line	Variance in word length in the messages
14.	Number of words in the message's body	Fraction of emails with attachments
15.	Number of characters in the message's subject line	Fraction of emails with replies or forwards
16.	Number of characters in the message's body	

To determine which of these features best distinguished viral and normal email behavior, we used feature selection to choose a subset of these features which empirically were most predictive of viruses. We employed a method discussed by Shawe-Taylor and Cristianini [2004] that finds the directions (*i.e.*, combinations of features) with maximal covariance with the labels and we selected the dominant feature representative of that direction in a greedy fashion. Using this feature selection, we winnowed the set of features used by our model down to the following seven features which we used to construct our detector: *i*) presence of HTML in the message, *ii*) number of files attached to the email, *iii*) presence of binary attachments, *iv*) fraction of emails with attachments, *v*) frequency of emails sent in the window, *vi*) average number of words in the message bodies, and *vii*) variance in the number of words in the message bodies These features provide strong indicators for the behavior of a mass-mailing virus primarily focusing on the presence of executable attachments, the frequency of sending messages, and repetition in the email content, which aligned well with our intuition about the characteristics of viruses. Based on these features, each message was represented to our virus detection system as a seven-dimensional vector.

Our detector used a multi-tiered approach to identify compromised hosts attempting to propagate their infection via email. The first stage in detection was a novelty detection technique called a *one-class support vector machine* (SVM), which can identify messages that significantly deviate from the normal data; *i.e.*, anomalous messages that are uncharacteristic of the user's normal behavior. Importantly, unlike the usual classification setting (see Chapter 2.2.4), a novelty detector learns by only observing normal messages. This property made the novelty detection paradigm well-suited to our setting since the normal behavior for a user was assumed to be (semi)-stable and non-adversarial whereas the behavior of different viruses may differ dramatically and future novel viruses could be designed specifically to deviate from the viral characteristics learned by our model. However, a pure novelty detection paradigm also has drawbacks—instead of learning specific viral characteristics it is only able to identify anomalous ones, which may not entirely coincide. As a result, we found that to have a reasonable detection rate, the one-class SVM had to have an unreasonably high false positive rate for a practical filter; *i.e.*, its ROC curve was unacceptably low. This led us to add a second stage into our filter.

Instead of using pure novelty detection, we instead used the one-class SVM to detect suspicious user behavior and then used a second layer of classification to determine whether or not a suspicious message was viral. This two-stage architecture allowed us to employ an extremely sensitive novelty detector with a low false negative rate (but high false positive rate) then correct most of the false positives by classifying the suspicious messages it identified as either viral or innocuous with a (two-class) naive Bayes classifier. In contrast to the novelty detector, the naive Bayes classifier was a per-user model capturing each individual user's email behavior. Thus, after an email was deemed suspicious by the novelty detector, a personalized model compared the email's characteristic to that user's previous behavior and to that of known viruses. We found the combined classification performance of this two-stage detection architecture surpassed the accuracy of either detector by itself as summarized in Table 1.1.

In the final stage of detection, messages deemed to be viral by our naive Bayes classifier were used to make a quarantine decision building on strategies by Williamson [2002] to throttle the spread of viruses. If sufficiently many messages in the recent past were deemed to be viral the machine would be quarantined until an administrator could disinfect it. Our

Experiment	'Novel' Email Virus Tested					
	BubbleBoy	Bagle.F	Netsky.D	Mydoom.U	Mydoom.M	Sobig.F
SVM Only						
Num. False Positives	198	219	219	215	222	222
Num. False Negatives	0	1	0	0	0	4
Num. Correctly Classified	1201	1179	1180	1184	1177	1173
% False Positives	16.50	18.25	18.25	17.92	18.50	18.50
% False Negatives	0.00	0.50	0.00	0.00	0.00	2.01
% Total Accuracy	85.85	84.27	84.35	84.63	84.13	83.85
Naive Bayes Only						
Num. False Positives	33	17	17	17	20	17
Num. False Negatives	8	4	4	4	4	5
Num. Correctly Classified	1358	1378	1378	1378	1375	1377
% False Positives	2.75	1.42	1.42	1.42	1.67	1.42
% False Negatives	4.02	2.01	2.01	2.01	2.01	2.51
% Total Accuracy	97.07	98.50	98.50	98.50	98.28	98.43
Two-Layer Model						
Num. False Positives	9	10	10	10	12	10
Num. False Negatives	8	4	4	4	4	5
Num. Correctly Classified	1382	1385	1385	1385	1383	1384
% False Positives	0.75	0.83	0.83	0.83	1.00	0.83
% False Negatives	4.02	2.01	2.01	2.01	2.01	2.51
% Total Accuracy	98.78	99.00	99.00	99.00	99.00	98.93

Table 1.1: Evaluation results of the accuracy of our virus detector against a number of email-bourne viruses (see Nelson [2005] for a detailed explanation of these results). Each experiment was repeated three times: first with only the one-class SVM, then using only a naive Bayes parametric classifier, and finally with the two-stage system. We report the number of false positives, false negatives, and correctly classified emails. The percentage of false positives/negatives is the percent of the normal/viral email misclassified.

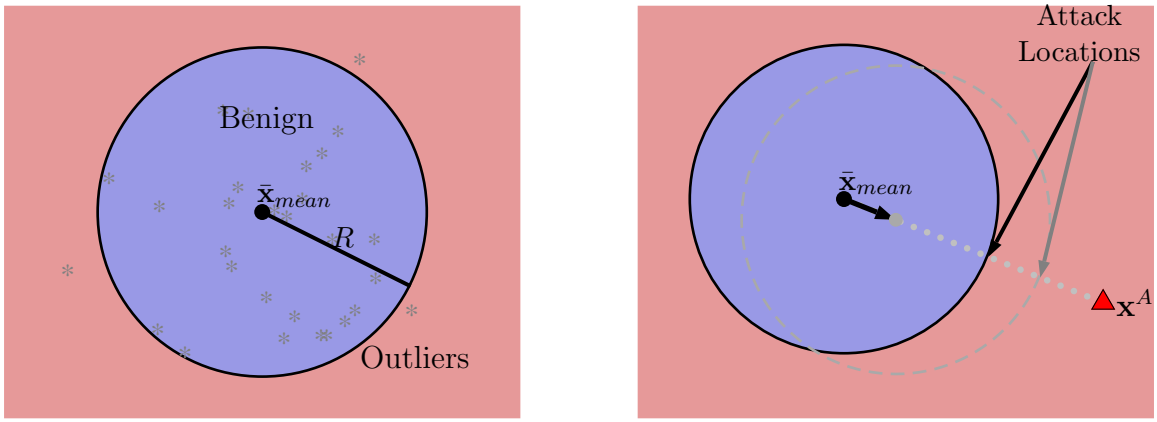
thresholding module was designed to mitigate the effect of false positives but at the cost of introducing some additional false negatives during the initial period of infection. Our quarantine strategy applied a threshold to the percentage of emails classified as infected over a sliding window of the last ten messages; if that threshold was exceeded, it would be possible to report, with high confidence, that a machine was infected and quarantine it. This approach allowed our detector to significantly reduce the virus' ability to propagate (and thus stymied their purpose) while further reducing the impact on normal users as is detailed in Sewani [2005].

In designing our virus detection system, my colleagues and I attempted to anticipate and prevent future virus outbreaks. By targeting the principal behaviors of fast-spreading email viruses (need to propagate quickly, need to send executable attachments, *etc.*), our detection system was designed to be robust against superfluous changes to viral behavior meant to confuse the detector without altering the actual effect of the virus. Further, by using two-stage classification, we hoped to make the detector more difficult to circumvent since an evading virus would have to navigate successfully through two layers of detection. However, while our system proved to be effective in detecting observed email virus outbreaks, it is again unclear if this approach could have effectively detected viruses designed to thwart it. Our hope was that a virus would have to significantly degrade its own objectives to evade detection (*e.g.*, a virus may slow its spread but, in doing so, it would defeat its own purpose) but we were unable to verify how effectively a virus could evade our system. In designing a two-layer detection system with a non-linear novelty detector, the resulting detector was difficult to interpret; *i.e.*, it was unclear what rules the detector had constructed to flag viruses and whether those rules had *blind spots*. Further, our multi-stage architecture was less robust than we had intended—rather than having to evade all the stages, a virus would only need to evade any single one. In retrospect, a better design for multiple detectors would be to treat each as an expert and aggregate their predictions as is discussed in Chapter 3.6. Finally, in designing our system, we never considered that our training data may be contaminated by malicious data—this oversight spawned my first project specifically addressing *adversarial learning*.

1.3.3 Hypersphere Model

In continuing to explore virus detection, I began investigating how vulnerable our learning algorithm was to adversarial contamination. The threat of an adversary systematically misleading our outlier detector led me to construct a theoretical model for analyzing the effect of contamination on our learning approach to virus detection. In my Master's Thesis [Nelson, 2005], I analyzed a simple algorithm for outlier detection based on bounding the normal data in a mean-centered hypersphere of fixed radius as depicted in Figure 1.2(a). I analyzed this detector instead of the one-class SVM primarily because the hypersphere is easier to analyze and I hoped the analysis used on it could be extended to hyperplane classifiers (like the one-class SVM) although these extensions have not yet been pursued.

In the hypersphere model, the novelty detector is a mean-centered hypersphere of fixed radius R (possibly in a kernel-space). This novelty detector uses a *bootstrapping retraining* policy—only adding points classified as normal to the training set while anomalous data points are discarded. Further, the *data points* in the training set are never removed; *i.e.*, there is no *aging* of data. I also made strong conservative assumptions about the attacker to



(a) Hypersphere Outlier Detection

(b) Attack on a Hypersphere

Figure 1.2: Depictions of the concept of hypersphere outlier detection and the vulnerability of naive approaches. **(a)** A bounding hypersphere centered at $\bar{\mathbf{x}}_{mean}$ of fixed radius R is used to encapsulate the empirical support of a distribution by excluding outliers beyond its boundary. Samples from the 'normal' distribution are indicated by $*$'s with three outliers on the exterior of the hypersphere. **(b)** How an attacker with knowledge about the state of the outlier detector can shift the outlier detector toward the goal $\mathbf{x}^A$. It will take several iterations of attacks to sufficiently shift the hypersphere before it encompasses $\mathbf{x}^A$ and classifies it as benign.

bound the minimal amount of effort he requires to be successful. I assumed the attacker is omnipotent—he knows the state of the novelty detector, the policies of the novelty detector, and how the novelty detector will change on retraining. I also assumed that the attacker could control *all* training data once his attack commenced.

For this basic model for novelty detection, I analyzed a contamination scenario whereby the attacker poisons the learning algorithm to pervert its ability to adapt into a tool the adversary uses to accomplish his objective. The objective I considered was that the adversary wants the novelty detector to misclassify a malicious target point $\mathbf{x}^A$ as a normal instance. However, the initial detector correctly classifies $\mathbf{x}^A$ as malicious so the adversary must manipulate the learner to achieve his objective. Initially, the attacker's target point $\mathbf{x}^A$ is located a distance D radii from the side of the hypersphere (or a total distance of $R\,(D+1)$ from its initial center), the initial hypersphere was trained using N initial benign data points, and the adversary has M total attack points to use during the attack which takes place over the course of T retraining iterations of the hypersphere model. The purpose of studying this simple attack scenario was to quantify the relationship between the attacker's effort (*i.e.*, M, the number of attack points required) and the attacker's impact (in terms of the number of radii D that the hypersphere is shifted) to gain a better understanding of the effectiveness of data contamination on learning agents.

Based on the assumptions made about the attacker's omnipotence and control of the training data, constructing optimal attacks for this model was straightforward. The optimal attacker can maximally displace the bounding hypersphere towards $\mathbf{x}^A$ by inserting the *attacks points* near the boundary along the line between the mean of the current hypersphere

and $\mathbf{x}^A$; *i.e.*, at the ℓ_2-projection of $\mathbf{x}^A$ onto the hypersphere. This attack strategy is depicted in Figure 1.2(b). The only question that remained was how to allocate the M attack points among the T retraining iterations, which I showed to be equivalent to a center-of-mass problem where T blocks of length $2R$ are stacked to maximize their extent beyond the edge of a precipice. Here the top t blocks have a total mass of M_t and the stack has a total mass of M with an additional point mass N on the outer edge of the top block. The optimal allocation of mass amongst these blocks must satisfy the following conditions:

$$\begin{aligned} M_0 &= N \\ \frac{M_{t-1}}{M_t} &= \frac{M_t}{M_{t+1}} \quad \forall\, 1 \leq t < T \\ M_T &= M + N \ . \end{aligned}$$

By relaxing the integer constraints on the M_t, I showed that the optimal real-valued solution is given by

$$M_t^* = N^{\left(\frac{T-t}{T}\right)} \cdot (M+N)^{\left(\frac{t}{T}\right)} \ .$$

This also yielded the following bound on the total number of attack points M required to shift the hypersphere by a distance of D radii over T iterations:

$$\begin{aligned} M &\geq N\left(1 - \frac{D}{R \cdot T}\right)^{-T} - N \\ &\geq N\left(\exp\left(\frac{D}{R}\right) - 1\right) \ . \end{aligned}$$

This bound was a positive result in that it showed that trimmed means are hard to poison as they accumulate training data in an online fashion. Thus, because of the bootstrap retraining and the retention of *all* old data, the mean-centered hypersphere was shown to be difficult to displace by large distances in that the required effort M is exponential in the desired displacement D. However, this result also indicated that such a retraining process becomes less adaptable to regular distributional shifts as more data accumulates, which reduces the utility of such a model considerably. This model lacked a realistic policy for retraining and makes the unrealistic assumption that the attacker controls *all* data during the attack. Nonetheless, this work served as a foundation for analyzing repeated contamination games in which the adversary attempts to poison a filter over many retraining iterations and Kloft and Laskov [2010] extended this model by considering more realistic retraining policies and a more realistic setting for the attack. Further, this early work on contamination models heavily influenced my subsequent approach to the adversarial learning framework that I describe in the remainder of this dissertation.

1.4 Dissertation Organization

The remainder of this dissertation is organized into three parts. In the first part, I present the background and foundational materials for this work. In the next chapter, I present a synopsis of machine learning and introduce my notation. Then in Chapter 3, I introduce a framework for assessing the security properties of learning agents, I present the taxonomy of attacks against learners, and I categorize and discuss the prior work within the context of

this framework. I also apply the framework to motivate two studies of practical applications of learning algorithms.

In the second part, I investigate two practical attacks against learning systems and corresponding defenses based on the framework. The first is a spam filter called SpamBayes that I investigate in Chapter 4. I show that the SpamBayes filter is vulnerable to attack messages that contaminate its training data causing it to subsequently misclassify normal messages as spam, but I also demonstrate that a simple *data sanitization* technique can effectively detect and remove attack messages with a minimal impact on the filter's performance. In Chapter 5, I study a class of data poisoning strategies against a second learning system—a PCA-based anomaly detector designed to identify network-wide DoS attacks in a backbone communication network. Again, I show that this class of algorithms is susceptible to poisoning, but in this case, I show that an alternative algorithm based on a robust variant of PCA is able to substantially mitigate the effect of the poisoning.

In the final part of the dissertation, I explore the near-optimal evasion problem for the family of convex-inducing classifiers. I apply the framework to a theoretical model of classifier evasion in Chapter 6. I generalize the near-optimal evasion framework of Lowd and Meek [2005b] to the broader family of convex-inducing classifiers and explore algorithms for evading these classifiers based on the family of ℓ_p costs. In the final chapter, Chapter 7, I conclude with a summary of the contributions of this dissertation and discuss important themes and open questions for the field of adversarial learning in security-sensitive domains. Below, I outline the primary contributions I make in this dissertation.

1.4.1 Contributions

In this dissertation, I present a systematic approach for identifying and analyzing threats against a machine learning system. I examine a number of real-world learning systems, assess their vulnerabilities, demonstrate real-world attacks against their learning mechanism, and propose defenses that can successfully mitigate the effectiveness of such attacks. In doing so, I provide machine learning practitioners with a systematic methodology for assessing a learner's vulnerability and developing defenses to strengthen their system against such threats. Additionally, I also examine and answer theoretical questions about the limits of adversarial contamination and classifier evasion.

My first major contribution is a central framework for categorizing and describing potential threats against a learning system (Chapter 3). This framework provides a taxonomy of attacks, which divides threats along three axes. These axes describe the fundamental characteristics of attacks that transcend domain-specific differences to elicit commonalities among attacks in very different problem domains. I show that a security threat can also be modeled as a game between the adversary and the learner, in which the characteristics of the threat define the nature of the game. Each of these games engenders potential limits on the adversary (and learner) and allow a security analyst to assess the learner's vulnerability. I identify plausible models for adversarial contamination and evasion of the learning algorithm. Finally, the framework plays an essential role in motivating the practical attacks on two realistic learning systems that are my next contribution.

The second principal contribution I make in this dissertation is a systematic investigation of attacks on real-world learning systems—first a spam filter in Chapter 4 then a

network anomaly detector in Chapter 5. For both of these system, I assess potential attacks against the learner based on the framework. I identify plausible objectives for attackers, construct a threat model, and investigate how effectively the attacker can achieve their objective in several different scenarios, which provide different levels of information to the adversary. By examining the adversary's impact in these different settings, I quantified attacks ranging from a worst-case omnipotent adversary to more realistic information and resource constrained opponents. For both learning systems, I demonstrate that realistic adversary's can have a devastating impact on the performance of these systems by making small adversarial alterations to the training data.

My third contribution is the design and analysis of defenses that I show can substantially mitigate attacks on real-world learning systems. The first is a data sanitization technique that I propose as a method of filtering contaminated training data for spam filtering (Chapter 4.4). The technique I propose estimates the impact each message has on the filter and excludes messages whose estimated impact is exceedingly damaging to the filter's performance. I show that this simple sanitization method is extraordinarily effective in preventing some attacks against the learning algorithm. The second technique is an alternative learning method for the network anomaly detection setting. I adapt a technique from robust statistics that was specifically designed to behave well under adversarial contamination. I show that the alternative method successfully mitigates the effectiveness of the attacks and outperforms the original detector under even small amounts of adversarial contamination. By constructing successful defenses against attacks, I establish methods for hardening vulnerable learning systems against potential threats and assess the limits of these defenses.

Finally, my fourth contribution is a generalization of a theoretical framework for assessing classifier vulnerability to evasion (Chapter 6). I build on the setting proposed by Lowd and Meek [2005b] for quantifying the difficulty for an adversary to programmatically find a classifier's most desirable blind spots in terms of the query complexity required by the adversary to find such an evading instance that is near-optimal according to some notion of adversarial cost. In my investigation, I show that near-optimal classifier evasion is possible for the family of convex-inducing classifiers with respect to weighted ℓ_1 costs. In doing so, I also demonstrate that the near-optimal evasion problem is generally computationally easier than reverse-engineering a classifier. I also examine the broader family of ℓ_p costs and present cases in which the convex-inducing classifiers cannot be efficiently evaded.

Chapter 2

Background and Notation

In this section, I establish the general notation I use throughout the remainder of this dissertation and introduce the basic foundation of machine learning that this dissertation builds upon. Readers generally familiar with this field can read this section cursorily to understand my notation. For a more thorough treatment of machine learning, the reader should refer to a text such as Hastie et al. [2003] or Vapnik [1995].

2.1 Notation and Terminology

Throughout this dissertation, I consistently use the following mathematical notation.

First-Order Logic: The notation $a \wedge b$ denotes the logical *conjunction* of a and b, $a \vee b$ denotes the logical *disjunction* of a and b, and $\neg a$ is the logical *negation* of a. I use the symbols $\forall$ and $\exists$ for universal and existential quantification, respectively. I denote predicates as functions such as $p(\cdot)$ which evaluates to true if and only if it's input exhibits the property represented by the predicate. The special *identity predicate* $\mathrm{I}[a]$ is true if and only if a is true. Also, for convenience, I overload this notation for the *indicator function*, which instead maps to $\{0, 1\}$ rather than $\{\text{false}, \text{true}\}$.

Sets: A *set*, or a collection of objects, is denoted using blackboard bold characters such as $\mathbb{X}$; the set with no elements is the *empty set*, $\emptyset$. However, when referring to the *entire* set or universe of a particular kind of object, I use calligraphic script characters such as $\mathcal{X}$ to distinguish it from sets $\mathbb{X} \subset \mathcal{X}$. To group a collection of objects as a set I use curly braces such as $\{a, b, c\}$. To specify set membership I use $x \in \mathbb{X}$, and to explicitly enumerate the elements of a set I use the notation $\mathbb{X} = \{x_1, x_2, \ldots, x_N\}$ for a finite set and $\mathbb{X} = \{x_1, x_2, \ldots\}$ for an infinite set. I use $\mathbb{Y} \subset \mathbb{X}$ to denote that $\mathbb{Y}$ is a *subset* of $\mathbb{X}$. For finite sets, I use the notation $|\mathbb{X}|$ to denote the size of $\mathbb{X}$. I use $\mathbb{X} \cup \mathbb{Y}$ to denote the *union* of two sets, $\mathbb{X} \cap \mathbb{Y}$ to denote their *intersection*, and $\mathbb{X} \setminus \mathbb{Y} \triangleq \{x \in \mathbb{X} \wedge x \notin \mathbb{Y}\}$ to denote the *set difference* of elements in $\mathbb{X}$ but not in $\mathbb{Y}$. To qualify the elements within a set, I use the notation $\mathbb{X} = \{x \mid A(x)\}$ to denote a set of objects that satisfy the predicate $A(\cdot)$. I also use function $\mathrm{I}_{\mathbb{X}}[\cdot]$ to denote the *set indicator function* for $\mathbb{X}$; *i.e.*, $\mathrm{I}_{\mathbb{X}}[x] \triangleq \mathrm{I}[x \in \mathbb{X}]$ (again I overload this function to map to $\{0, 1\}$ for convenience).

Integers and reals: Common sets include the set of all integers $\mathfrak{Z}$ and the set of all real numbers $\Re$. A special subset of the integers is the natural numbers $\aleph = \{z \in \mathfrak{Z} \mid \ z > 0\}$. Similarly, special subsets of the reals are the positive reals $\Re^{+} = \{r \in \Re \mid \ r > 0\}$ and the non-negative reals $\Re^{0+} = \{r \in \Re \mid \ r \geq 0\}$.

Indexed Sets: To order the elements of a set, I use an *index set* as a mapping to each element. For a finite indexable set, I use the notation $\left\{x^{(i)}\right\}_{i=1}^{N}$ so that the sequence of N objects are indexed by $\{1, \ldots, N\}$. More generally, a set indexed by the $\mathbb{I}$ is denoted $\left\{x^{(i)}\right\}_{i \in \mathbb{I}}$. An infinite set can be indexed by using $\aleph$ or $\Re$ as the index set depending on its cardinality.

Multi-dimensional sets: Sets can also be coupled to describe multi-dimensional objects or tuples which I denote with a (lowercase) bold character such as $\mathbf{x}$. An *ordered pair* $\langle x, y \rangle \in \mathbb{X} \times \mathbb{Y}$ is a pair of objects $x \in \mathbb{X}$ and $y \in \mathbb{Y}$. This ordered pair belongs to the Cartesian product of the sets $\mathbb{X}$ and $\mathbb{Y}$ defined as the set of all such ordered pairs: $\mathbb{X} \times \mathbb{Y} \triangleq \{ \langle x, y \rangle \mid \ x \in \mathbb{X} \ \wedge y \ \in \mathbb{Y}\}$. A *n-tuple* is an ordered list of n objects from n sets: $\langle x_1, x_2, \ldots, x_n \rangle \in \bigtimes_{i=1}^{n} \mathbb{X}_i$ where the generalized Cartesian product $\bigtimes_{i=1}^{n} \mathbb{X}_i \triangleq \mathbb{X}_1 \times \mathbb{X}_2 \times \ldots \times \mathbb{X}_n = \{ \langle x_1, x_2, \ldots, x_n \rangle \mid \ x_1 \in \mathbb{X}_1 \ \wedge x_2 \ \in \mathbb{X}_2 \ \wedge \ \ldots \ \wedge \ x_n \in \mathbb{X}_n\}$ is the set of all such n-tuples. Here, the *dimension* of the space or the objects in it is n and the function $\dim(\cdot)$ returns the dimension of an object. When each element of a n-tuple belongs to a common set $\mathbb{X}$, the generalized Cartesian product is denoted with exponential notation as $\mathbb{X}^n \triangleq \bigtimes_{i=1}^{n} \mathbb{X}_i$; *e.g.*, the Euclidean space $\Re^n$ is the n-dimensional real-valued space.

Vectors: For my purposes *vector* is a special case of ordered n-tuple that I represent as with a (usually lowercase) bold character such as $\mathbf{v}$; unlike general tuples, vectors are associated with an addition and a scalar multiplication operation. For an n-vector $\mathbf{v}$ with elements in the set $\mathbb{X}$, $\mathbf{v} \in \mathbb{X}^n$. The i^{th} element (*coordinate*) of $\mathbf{v}$ is a scalar denoted by $v_i \in \mathbb{X}$ where $i \in \{1, 2, \ldots, n\}$. Special real-valued vectors include the all ones vector $\mathbf{1} = \langle 1, 1, \cdots, 1 \rangle$, the all zeros vector $\mathbf{0} = \langle 0, 0, \cdots, 0 \rangle$, and the coordinate/basis vector $\mathbf{e}^{(d)} \triangleq \langle 0, \ldots, 1, \ldots, 0 \rangle$ which has 1 only in its d^{th} coordinate and 0 elsewhere.

Sequences of Objects: I differentiate sequenced objects from vectors by using the notation $x^{(t)}$ to denote the t^{th} object in a sequence. This is to avoid confusion in referring to a sequence of multi-dimensional data. Here $\mathbf{x}^{(t)}$ refers to the t^{th} n-dimensional vector in a sequence, $x_i^{(t)}$ refers to the i^{th} element of it, and x_i^t is the t^{th} power of x_i.

Vector Spaces: A *vector space* is a set of vectors that can be added or multiplied by a scalar; *i.e.*, the space is closed under vector addition and scalar multiplication operations that obey associativity, commutativity, and distributivity and has an additive and multiplicative identity as well as additive inverses. For example, the Euclidean space $\Re^n$ is a vector space for any $n \in \aleph$. A *convex set* $\mathbb{C} \subset \mathbb{X}$ is a subset of a vector space with the property that $\forall \alpha \in [0, 1] \quad x, y \in \mathbb{C} \ \Rightarrow \ (1 - \alpha)x + \alpha y \in \mathbb{C}$; *i.e.*, all convex combinations of any $x \in \mathbb{C}$ and $y \in \mathbb{C}$ are also in $\mathbb{C}$. A vector space $\mathbb{X}$ is a *normed vector space* if it is associated with a *norm function* $\|\cdot\| : \mathbb{X} \rightarrow \Re$ on the space such that *i*) there is a zero element 0 that satisfies $\|x\| = 0 \iff x = 0$, *ii*) for any scalar α, $\|\alpha x\| = |\alpha| \, \|x\|$, and

iii) the triangle inequality holds: $\|x+y\| \leq \|x\| + \|y\|$. A common family of norms are the ℓ_p norms defined as

$$\|\mathbf{x}\|_p \triangleq \sqrt[p]{\sum_{i=1}^{n} |x_i|^p} \tag{2.1}$$

for $p \in \Re^+$.

Matrices: I represent matrices using a (usually uppercase) bold character such as $\mathbf{A}$. A *matrix* is a multi-dimensional object with two indices. For an $M \times N$-matrix with elements in the set $\mathbb{X}$, $\mathbf{A} \in \mathbb{X}^{M \times N}$, and I overload $\dim(\cdot)$ to return the tuple $\langle M, N \rangle$; the number of rows and columns of the matrix. The $\langle i,j \rangle^{\text{th}}$ element of $\mathbf{A}$ is denoted by $A_{i,j} \in \mathbb{X}$ where $i \in \{1, 2, \ldots, M\}$ and $j \in \{1, 2, \ldots, N\}$. The full matrix can then be expressed element-wise using the bracket notation:

$$\mathbf{A} = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,N} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{M,1} & A_{M,2} & \cdots & A_{M,N} \end{bmatrix} .$$

As suggested by this notation, the first index of the matrix refers to its *row* and the second refers to its *column*. Each row and each column are themselves vectors and are denoted by $\mathbf{A}_{i,\bullet}$ and $\mathbf{A}_{\bullet,j}$ respectively. I also use the bracket notation $[\cdot]_{i,j}$ to refer to the $\langle i,j \rangle^{\text{th}}$ element of a matrix-valued expression. Special matrices include the identity matrix $\mathbf{I}$, with 1's along its diagonal and 0's elsewhere, and the zero matrix $\mathbf{0}$ with zero in every element. The transpose of an $M \times N$-matrix is an $N \times M$-matrix denoted as $\mathbf{A}^\top$ and defined as $\left[\mathbf{A}^\top\right]_{i,j} = \mathbf{A}_{j,i}$.

Vector/Matrix Multiplication: Here I consider vectors and matrices whose elements belong to a set $\mathbb{X}$ with pairwise multiplication (*e.g.*, $\mathfrak{Z}$, $\Re$). For the purpose of matrix multiplication, I represent an N-vector as an $N \times 1$ matrix for convenience. The *inner product* between two vectors $\mathbf{v}$ and $\mathbf{w}$ ($\dim(\mathbf{v}) = \dim(\mathbf{w})$) is a scalar denoted by $\mathbf{v}^\top \mathbf{w} = \sum_{i=1}^{N} v_i \cdot w_i$. The *outer product* between M-vector $\mathbf{v}$ and N-vector $\mathbf{w}$ is an $M \times N$-matrix denoted by $\mathbf{v}\mathbf{w}^\top$ with elements $\left[\mathbf{v}\mathbf{w}^\top\right]_{i,j} = v_i \cdot w_j$. The product between an $M \times N$-matrix $\mathbf{A}$ and an N-vector $\mathbf{w}$ is denoted $\mathbf{A}\mathbf{w}$ and defined as the M-vector of inner products between the i^{th} row $\mathbf{A}_{i,\bullet}$ and the vector $\mathbf{w}$; *i.e.*, $\langle \mathbf{A}\mathbf{w} \rangle_i = \mathbf{A}_{i,\bullet}^\top \mathbf{w}$. It follows that $\mathbf{v}^\top \mathbf{A} \mathbf{w}$ is a scalar defined as $\mathbf{v}^\top \mathbf{A} \mathbf{w} = \sum_{i,j} v_i \cdot A_{i,j} \cdot w_j$. The *matrix product* between an $K \times M$-matrix $\mathbf{B}$ and an $M \times N$-matrix $\mathbf{A}$ is an $K \times N$-matrix denoted as $\mathbf{B}\mathbf{A}$ whose $\langle i,j \rangle^{\text{th}}$ element is the inner product between the i^{th} row of $\mathbf{B}$ and the j^{th} column of $\mathbf{A}$; *i.e.*, $[\mathbf{B}\mathbf{A}]_{i,j} = \mathbf{B}_{i,\bullet}^\top \mathbf{A}_{\bullet,j}$.

I also consider the *Hadamard (element-wise) product* of vectors and matrices which I denote with the $\odot$ operator. The Hadamard product of vectors $\mathbf{v}$ and $\mathbf{w}$ ($\dim(\mathbf{v}) = \dim(\mathbf{w})$) is a vector defined as $\langle \mathbf{v} \odot \mathbf{w} \rangle_i \triangleq v_i \cdot w_i$. Similarly, the Hadamard product of matrices $\mathbf{A}$ and $\mathbf{B}$ ($\dim(\mathbf{A}) = \dim(\mathbf{B})$) is a matrix defined as $[\mathbf{A} \odot \mathbf{B}]_{i,j} \triangleq A_{i,j} \cdot B_{i,j}$.

Functions: I denote a function using regular italic font; *e.g.*, g. However, for special named functions (such as log and sin) I use the non-italicized Roman font. A set is a

mapping from its *domain* $\mathbb{X}$ to its *codomain* $\mathbb{Y}$; $g : \mathbb{X} \to \mathbb{Y}$. To apply g to x, I use the usual notation $g(x)$; $x \in \mathbb{X}$ is the argument and $g(x) \in \mathbb{Y}$ is the value of g at x. I also use this notation to refer to parameterized objects but it this case, I will name the object according to the type of object. For instance, $\mathbb{B}^C(g) \triangleq \{x \mid g(x) < C\}$ is a set parameterized by the function f called the C-ball of g and so I call attention to the fact that this object is a set by using the set notation $\mathbb{B}$.

Families of functions: A family of functions is a set of functions, for which I extend the previous concept of multi-dimensional sets. Functions can be defined as tuples of infinite length—instead of indexing the tuple with natural numbers, it is indexed by the domain of the function; *e.g.*, the reals. To represent the set of all such functions, I use the generalized Cartesian product over an index set $\mathbb{I}$ as $\bigtimes_{i \in \mathbb{I}} \mathbb{X}$ where $\mathbb{X}$ is the codomain of the functions. For instance the set of all real-valued functions is $\mathcal{G} = \bigtimes_{x \in \Re} \Re$; *i.e.*, every function $g \in \mathcal{G}$ is a mapping from the reals to the reals: $g : \Re \to \Re$. I also consider special subsets such as the set of all continuous real-valued functions $\mathcal{G}^{(\text{continuous})} = \{g \in \mathcal{G} \mid \textit{continuous}(g)\}$ or the set of all convex functions $\mathcal{G}^{(\text{convex})} = \{g \in \mathcal{G} \mid \forall\, t \in [0,1] \;\; g(tx + (1-t)\,y) \leq tg(x) + (1-t)\,g(y)\}$. Particularly, I use the family of all classifiers in a D-dimensional space in Chapter 6. This family is the set of functions mapping $\Re^D$ to the set $\{'-','+'\}$ and denoted by $\mathcal{F} \triangleq \bigtimes_{x \in \Re^D} \{'-','+'\}$.

Optimization: Learning theory draws heavily from mathematical optimization. Optimization typically is cast as finding the *best* object x from a set $\mathcal{X}$ in terms of finding a minimizer of an objective function $f : \mathcal{X} \to \Re$:

$$x^* \in \operatorname*{argmin}_{x \in \mathcal{X}} [f(x)]$$

where $\operatorname{argmin}[\cdot]$ is a mapping from the space of all objects $\mathcal{X}$ to a subset $\mathbb{X}' \subset \mathcal{X}$ which is the set of all objects in $\mathcal{X}$ that minimize f (or equivalently maximize $-f$). Optimizations can also be restricted to obey a set of *constraints*. When specifying an optimization with constraints, I use the following notation:

$$\begin{aligned} &\operatorname{argmin}_{x \in \mathcal{X}} [f(x)] \\ &\quad \text{s.t.} \qquad\quad C(x) \end{aligned}$$

where f is the function being optimized and C represents the constraints that need to be satisfied. Often there will be several constraints C_i that must be satisfied in the optimization.

Probability and Statistics: I denote a probability distribution over the space $\mathcal{X}$ by $P_{\mathcal{X}}$. It is a nonnegative function, which is defined on the subsets in a σ-field of $\mathcal{X}$ (*i.e.*, a set of subsets of $\mathcal{X}$ that is closed under complements and countable unions) and satisfies (*i*) $p_{\mathcal{X}}(\mathbb{A}) \geq 0$, (*ii*) $p_{\mathcal{X}}(\mathcal{X}) = 1$, and (*iii*) for pairwise disjoint subsets $\mathbb{A}^{(1)}$, $\mathbb{A}^{(2)}$, ... it yields $p_{\mathcal{X}}\left(\bigcup_i \mathbb{A}^{(i)}\right) = \sum_i \mathbb{A}^{(i)}$ (for a more thorough treatment, refer to Billingsley [1995]). A random variable drawn from distribution $P_{\mathcal{X}}$ is denoted by $X \sim P_{\mathcal{X}}$—notice that I do not use a special notation for the random variable but I make it clear in the text that they are random. The expected value of a random variable is denoted by $\mathrm{E}_{X \sim P_{\mathcal{X}}}[X] = \int x \mathrm{d}p_{\mathcal{X}}(x)$. The family of all probability distributions on $\mathcal{X}$ is denoted by $\mathcal{P}_{\mathcal{X}}$; as above, this is a family of functions.

2.2 Statistical Machine Learning

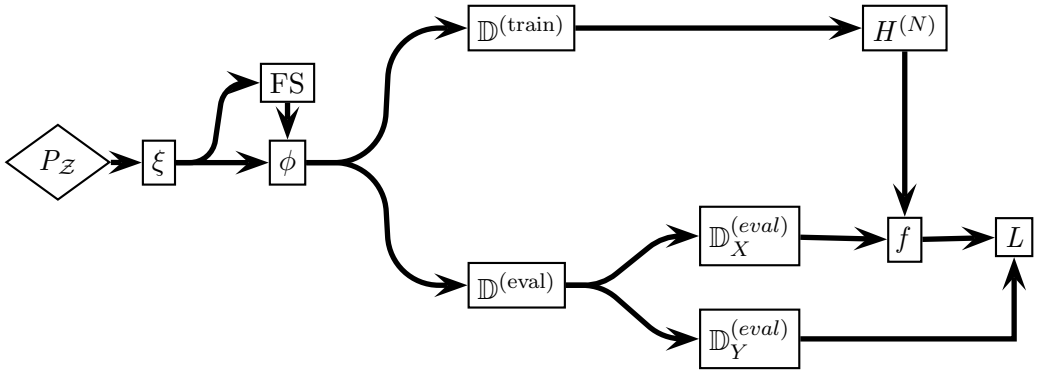
Machine learning encompasses a vast field of techniques that extract information from data as well as the theory and analysis relating to these algorithms. In describing the task of machine learning, Mitchell [1997] wrote,

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E

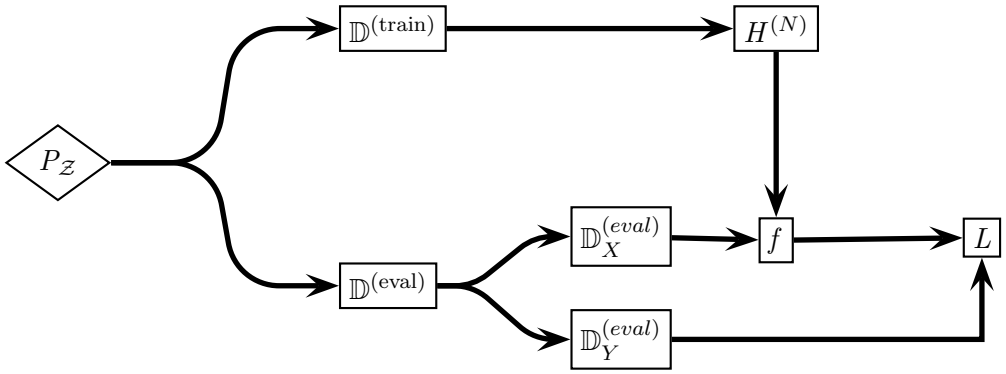
This definition encompasses a broad class of methods. Here, I present an overview of the terminology and mechanisms for a particular notion of learning that is often referred to as statistical machine learning. In particular, the notion of experience is cast as *data*, the task is to choose an *action* (or make a prediction/decision) from a set of possible actions/decisions, and the performance metric is a *loss function* that measures the cost the learner incurs for a particular prediction/action compared to the best or correct one. Figure 2.1 illustrates the data flow for learning in this setting: data $\mathbb{D}^{(\text{train})}$ drawn from the distribution $P_{\mathcal{Z}}$ is used by the learning procedure $H^{(N)}$ to produce a *hypothesis* (or classifier) f. This classifier is a function that makes predictions on a new set of data $\mathbb{D}^{(\text{eval})}$ (drawn from the same distribution) and is assessed according to the loss function L. The instance space $\mathcal{Z}$ is discussed in more detail below, but generally instances z are drawn from $\mathcal{Z}$ according to the distribution $P_{\mathcal{Z}}$ and serve to train and evaluate the classifier f. Also, Figure 2.1(a) additionally depicts the data collection phase of learning discussed briefly below. While measurement and feature selection are important aspects of the security of a learning algorithm, I do not focus on them in this dissertation.

Throughout, I only consider *inductive learning* methods, for which learning takes the form of generalizing from prior experiences. The method of induction requires an *inductive bias*, a set of (implicit) assumptions used to create generalizations from a set of observations. An example of an inductive bias is *Ockham's Razor*—preference for the simplest hypothesis that is consistent with the observations. Usually, the inductive bias of these methods is an implicit bias built into the learning procedure, but I do not discuss it further.

In this dissertation, I primarily focus on techniques from statistical machine learning that can be described as empirical risk minimization procedures. Below, I summarize these procedures and provide notation to describe them, but at a high level, empirical risk minimization procedures attempt to minimize the total loss incurred for each prediction made about the evaluation data, $\mathbb{D}^{(\text{eval})}$. Fundamentally, assuming stationarity in the data, minimizing the expected loss (or risk) on the training data is a surrogate to minimize loss on the evaluation data and, under the appropriate conditions, the error on the training data can be used to bound the generalization error [*cf.*, Vapnik, 1995, Chapter 1]. Underlying these results is the assumption of *stationarity* that the training data and evaluation data are both drawn from the same stationary distribution $P_{\mathcal{Z}}$ as depicted in Figure 2.1. Subsequently, I examine scenarios that violate this stationarity assumption and I evaluate the impact these violations have on the performance of learning methods. However, while I study the impact on performance of empirical risk minimizers, these violations would have similar effects on any learner based on stationarity, and further I verify that these violations have less of an impact on alternative empirical risk minimizers that were designed to be robust



(a) The complete learning framework.



(b) The learning framework with implicit data collection.

Figure 2.1: Diagrams depicting the flow of information through different phases of learning. **(a)** All major phases of the learning algorithm except for model selection. Here objects drawn from $P_{\mathcal{Z}}$ are parsed into measurements which then are used in the feature selector FS. It selects a feature mapping ϕ which is used to create training and evaluation datasets, $\mathbb{D}^{(\text{train})}$ and $\mathbb{D}^{(\text{eval})}$. The learning algorithm $H^{(N)}$ selects a hypothesis f based on the training data and its predictions are assessed on $\mathbb{D}^{(\text{eval})}$ according to the loss function L. **(b)** The training and prediction phases of learning with implicit data collection phases. These learning phases are the focus of this dissertation.

against distributional deviations. Thus, vulnerabilities are neither unique to empirical risk minimization procedures nor are they inherent to them, but rather guarding against these exploits requires learners designed to be resilient against violations in their assumptions. Of course, there is also a trade-off in this robustness and the effectiveness of the procedure, which I highlight in each chapter.

2.2.1 Data

Real-world objects such as emails or network packets occur in a space Ω of all such objects. Usually, applying a learning algorithm directly to real-world objects is difficult because the learner cannot parse the objects' structure or the objects may have extraneous elements that are irrelevant to the learner's task. Thus, these objects are transformed into a more amenable representation by a mapping from real-world abstractions (*e.g.*, objects or events) into a set of representative observations—the process of *measurement*. In this process, each real-world abstraction, $\omega \in \Omega$, is measured and represented to the learning algorithm as a composite object $x \in \mathcal{X}$. Typically there are D simple measurements of ω; the i^{th} measurement (or *feature*) x_i is from a space $\mathcal{X}_i$, and the composite representation (or data point) $\mathbf{x} \in \mathcal{X}$ is represented as a tuple $\langle x_1, x_2, \ldots, x_D \rangle$. The space of all such data points is $\mathcal{X} \triangleq \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_D$. Each feature is usually real-valued $\mathcal{X}_i = \Re$, integer-valued $\mathcal{X}_i = \mathfrak{Z}$, boolean $\mathcal{X}_i = \{\text{true}, \text{false}\}$, or categorical $\mathcal{X}_i = \{A_1, A_2, \ldots, A_k\}$. Formally, I represent the measurement process with the *measurement map* $\xi : \Omega \mapsto \mathcal{X}$. It represents the learner's view of the world.

Data collection is the application of a measurement map ξ to a sequence of N objects $\omega^{(1)}, \omega^{(2)}, \ldots, \omega^{(N)}$ resulting in an indexed set of N data points $\left\{x^{(i)}\right\}_{i=1}^{N} \subset \mathcal{X}^N$, which I refer to as a *dataset* and denote it by $\mathbb{D}$. The dataset represents a sequence of observations of the environment and serve as the basis for learner's ability to generalize past experience to future events or observations. Various assumptions are made about the structure of the dataset, but most commonly, the learner assumes the data points are independent and identically distributed. All the learning algorithms I investigate assume that the data is independently sampled from an unknown stationary distribution although with various degrees of dependence on this assumption.

Labels In many learning problems, the learner is tasked with learning to predict the unobserved state of the world based on its observed state. Thus, observations are partitioned into two sets. Those that are observed are the *explanatory variables* (also referred to as the input, predictor, or controlled variables) and the unobserved quantities to be predicted comprise the *response variables* (also referred to as the output or outcome variables). In the context of this dissertation and my focus on *classification*, I refer to the observed independent quantities as the data point (as discussed above) and to the dependent quantities as its *label*. The learner is expected to be able to predict the label for a data point having seen past instances of data points coupled with their labels. In this form, each datum consists of two paired components: a data point x from an *input space* $\mathcal{X}$ and a label y from a *response space* $\mathcal{Y}$. These paired objects belong to the Cartesian product: $\mathcal{Z} \triangleq \mathcal{X} \times \mathcal{Y}$. Instances are drawn from a joint distribution $P_{\mathcal{Z}}$ over this paired space.

In learning problems that include labels (*e.g.*, supervised or semi-supervised learning), the learner trains on a set of paired data from $\mathcal{Z}$. In particular, a *labeled dataset* is an

indexed set of N instances from $\mathcal{Z}$: $\mathbb{D} \triangleq \left\{z^{(1)}, z^{(2)}, \ldots, z^{(N)}\right\}$ where $z^{(i)} \in \mathcal{Z}$ is drawn from $P_{\mathcal{Z}}$. The indexed set of just the data points is $\mathbb{D}_X \triangleq \left\{x^{(1)}, x^{(2)}, \ldots, x^{(N)}\right\}$ and the indexed set of just the labels is $\mathbb{D}_Y \triangleq \left\{y^{(1)}, y^{(2)}, \ldots, y^{(N)}\right\}$. In the case that $\mathcal{X} = \mathcal{A}^D$ for some numeric set $\mathcal{A}$, the i^{th} data point can be expressed as a D-vector $\mathbf{x}^{(i)}$ and the data can be expressed as a $N \times D$ matrix $\mathbf{X}$ defined by $\mathbf{X}_{i,\bullet} = \mathbf{x}^{(i)}$. Similarly, when $\mathcal{Y}$ is a scalar set (*e.g.*, booleans, reals), $y^{(i)}$ is a scalar and the labels can be expressed as a simple N-vector $\mathbf{y}$.

Feature Selection Typically, measurement is only the first phase in the overall process of data extraction. After a dataset is collected, it is often altered in a process of *feature selection*. Feature selection is a mapping ϕ of the original measurements into a space $\hat{\mathcal{X}}$ of features[1]: $\phi_{\mathbb{D}} : \mathcal{X} \mapsto \hat{\mathcal{X}}$. Unlike the data-independent measurement mapping ξ, the *feature selection map* often is selected in a data-dependent fashion to extract aspects of the data most relevant to the learning task. Further, measurement often is an irreversible physical process whereas feature selection usually can be redone by reprocessing the original measurements. In many settings, one can retroactively alter the feature selection process by redefining the feature selection map and reapplying it to the measured data whereas it is impossible to make retroactive measurements on the original objects unless they are stored. However, for the purposes of this dissertation, I do not distinguish between the feature selection and measurement phases because the attacks I study target other aspects of learning. I merge them together into a single step and disregard $\hat{\mathcal{X}}$ except explicitly in reference to feature selection. I further discuss potential roles for feature selection in security-sensitive settings in Chapter 7.2.

2.2.2 Hypothesis Space

A learning algorithm is tasked with selecting a hypothesis that best supports the data. Here I consider the hypothesis to be a function f mapping from the data space $\mathcal{X}$ to the response space $\mathcal{Y}$; *i.e.*, $f : \mathcal{X} \to \mathcal{Y}$. Of course there are many such hypotheses. I assume f belongs to a family of all possible hypotheses $\mathcal{F}$. The family of all possible hypotheses (or *hypothesis space*) is most generally the set of all functions that map $\mathcal{X}$ onto $\mathcal{Y}$: $\mathcal{F} \triangleq \{f \mid f : \mathcal{X} \to \mathcal{Y}\}$. The hypothesis space $\mathcal{F}$ may be constrained by assumptions made about the form of the hypotheses. For instance, learners often only consider the space of generalized linear functions of the form $f^{\beta}_{\mathbf{a},b}(\mathbf{x}) \triangleq \beta\left(\mathbf{a}^{\top}\mathbf{x} + b\right)$ where $\beta : \Re \mapsto \mathcal{Y}$ is some mapping from the reals to the response space. For instance, in the case that $\mathcal{Y} = \{0, 1\}$, the function $\beta(x) = \mathrm{I}[x > 0]$ yields the family of all halfspaces on $\Re^D$ parameterized by $\langle \mathbf{a}, b \rangle$. In the case that $\mathcal{Y} = \Re$ the identity function $\beta(x) = x$ defines the family of linear functions on $\Re^D$ also parameterized by $\langle \mathbf{a}, b \rangle$.

2.2.3 The Learner

I describe the *learner* as a model that captures assumptions made about the observed data—the model provides limitations on the space of hypotheses, and perhaps provides

[1] In the literature, feature selection chooses a subset of the measurements ($\hat{\mathcal{X}} \subset \mathcal{X}$), and feature extraction creates composite features from the original measurements. I do not differentiate between these two processes and will refer to both as feature selection.

prior knowledge or preferences on these hypotheses (*e.g.*, a prior in a Bayesian setting or a regularizer in a risk minimization setting). That is, the model is a set of assumptions about the relationship between the observed data and the hypothesis space, but the model does not specify how hypotheses are selected—that is done by the training procedure. For example, consider a simple location estimation procedure for normally distributed data. The data model specifies that the data points are independently drawn from a unit-variance *Gaussian distribution* centered at an unknown parameter θ; *i.e.*, $X \sim \mathrm{N}(\theta, 1)$. However, both the mean and the median are procedures for estimating the location parameter θ both of which are consistent with the model. By distinguishing the model and the training procedure one can study different aspects of a learner's vulnerabilities.

2.2.4 Supervised Learning

The primary focus of this work will be analyzing the task of *prediction* in supervised learning. In the supervised learning framework, the observed data points are paired: $\mathbb{D} = \left\{\left\langle x^{(i)}, y^{(i)}\right\rangle\right\}$ where $x^{(i)} \in \mathcal{X}$ and $y^{(i)} \in \mathcal{Y}$—a predictor (input) variable and its response (output) variable. I assume the dataset is drawn from a joint distribution $P_{\mathcal{Z}}$ over the space $\mathcal{Z}$ that may also be denoted as $P_{\mathcal{X}\times\mathcal{Y}}$. The objective of prediction is to select an appropriate hypothesis; *i.e.*, a map $f : \mathcal{X} \to \mathcal{Y}$ predicting the response variable based on the observed predictor variable. The learner selects the *best* hypothesis $f^\dagger$ from a space of all possible hypotheses $\mathcal{F}$.

Given a hypothesis space, $\mathcal{F}$, the goal is to learn a classification hypothesis (classifier) $f^\dagger \in \mathcal{F}$ to minimize errors when predicting labels for new data, or if our model includes a cost function over errors, to minimize the total cost of errors. The cost function assigns a numeric cost to each combination of data instance, true label, and classifier label. The defender chooses a procedure $H^{(N)}$, or *learning algorithm*, for selecting hypotheses.

The learner is a mapping from a dataset $\mathbb{D} \in \mathcal{Z}^N$ to a hypothesis f in the hypothesis space: $H^{(N)} : \mathcal{Z}^N \to \mathcal{F}$. I use $H^{(N)} : \mathcal{Z}^N \to \mathcal{F}$ to denote a *training algorithm*; that is, a mapping from N training examples to some hypothesis f in the hypothesis space $\mathcal{F}$. If the algorithm has a randomized element I use the notation $H^{(N)} : \mathcal{Z}^N \times \Re \to \mathcal{F}$ to capture that fact that the hypothesis depends on a random element $R \sim P_{\Re}$.

I also consider asymptotic procedures; that is, the hypothesis generated by a training algorithm that takes an entire distribution $P_{\mathcal{Z}} \in \mathcal{P}_{\mathcal{Z}}$ as its input. An asymptotic procedure is denoted by $H : \mathcal{P}_{\mathcal{Z}} \to \mathcal{F}$. An asymptotic learning procedure is a mapping from an entire distribution over $\mathcal{Z}$ to a function in the hypothesis space: $H : \mathcal{P}_{\mathcal{Z}} \to \mathcal{F}$. The finite sample version of the learner, $H^{(N)}$ can be viewed as the asymptotic procedure applied to the empirical distribution function.

Training The process I describe here is *batch training*—the learner trains on a training set $\mathbb{D}^{(\text{train})}$ and is evaluated on an evaluation set $\mathbb{D}^{(\text{eval})}$. This setting can be generalized to an repeated process of *online training*, in which the learner continually re-trains on evaluation data after obtaining its labels (I return to this setting in Chapter 3.6). In a pure online setting, prediction and re-training occur every time a new data point is received. In the batch training setting (or in a single ply of online learning), the learner forms a hypothesis $f^\dagger$ based on the collected data $\mathbb{D}^{(\text{train})}$—the process known as *training*. A plethora of different

training procedures have been used in the supervised learning setting for (regularized) empirical risk minimization under a wide variety of settings. I will not detail these methods further, but instead introduce the basic setting for classification.

In a classification problem the response space is a finite set of labels each of which correspond to some subset of input space (although these subsets need not be disjoint). The learning task is to construct a *classifier* that can correctly assign these labels to new data points based on labeled training examples from each class. In a *binary classification* setting there are only two labels, '−' and '+'; *i.e.*, the response space is $\mathcal{Y} = \{'-', '+'\}$. Where mathematically convenient, I will use 0 and 1 in place of the labels '−' and '+'; *i.e.*, I will implicitly redefine the label y to be $\mathrm{I}['+' = y]$. In binary classification, I refer to the two classes as the *negative class* ($y = '-'$) and the *positive class* ($y = '+'$). The training set $\mathbb{D}^{(\text{train})}$ consists of labeled instances from both classes. I primarily focus on binary classification for security applications, in which a *defender* attempts to separate instances (*i.e.*, data points), some or all of which come from a malicious *attacker*, into harmful and benign classes. This setting covers many interesting security applications, such as host and network intrusion detection, virus and worm detection, and spam filtering. In detecting malicious activity, the positive class (label '+') indicates malicious *intrusion instances* while the negative class (label '−') indicates benign or innocuous *normal instances*. In Chapter 5, I also consider the *anomaly detection* setting, in which the training set only contains normal instances from the negative class.

Risk Minimization The goal of the learner is to find the *best* hypothesis f^* from the hypothesis space $\mathcal{F}$ that best predicts the target concept (according to some measure of correctness) on instances drawn according to the unknown distribution $P_{\mathcal{Z}}$. Ideally the learner is able to distinguish f^* from any other hypothesis $f \in \mathcal{F}$ based on the observed data $\mathbb{D}$ of data points drawn from $P_{\mathcal{Z}}$, but this is seldom realistic or even possible. Instead, the learner should choose the *best* hypothesis in the space according to some criteria for preferring one hypothesis over another—this is the *performance measure*. The measure can be any assessment of a hypothesis; in statistical machine learning, a common procedure is *empirical risk minimization* which is based on a loss function $L : \mathcal{Y} \times \mathcal{Y} \mapsto \Re^{0+}$. The learner selects a hypothesis $f^\dagger \in \mathcal{F}$ that minimizes the expected loss, or *risk*, over all hypotheses ($f^\dagger \in \operatorname{argmin}_{f \in \mathcal{F}} R(P_{\mathcal{Z}}, f)$) where the risk is given by

$$R(P_{\mathcal{Z}}, f) \triangleq \int_{\langle x,y \rangle \in \mathcal{Z}} L(y, f(x)) \, \mathrm{d}p_{\mathcal{Z}}(x, y) \ .$$

However, this minimization is also infeasible since the distribution $P_{\mathcal{Z}}$ is unknown. Instead, in the empirical risk minimization framework, the learner selects $f^\dagger$ to minimize the empirical risk on the dataset $\mathbb{D} \sim P_{\mathcal{Z}}$ defined as

$$\tilde{R}_N(f) = \frac{1}{N} \sum_{\langle x,y \rangle \in \mathbb{D}} L(y, f(x))$$

where $N = |\mathbb{D}|$. The practice of minimizing this surrogate for the true risk is known as *empirical risk minimization* [*cf.*, Vapnik, 1995].

Regularization The learner also should restrict the space of hypotheses $\mathcal{F}$. If the space of hypotheses is too expressive, there will be a hypothesis that fits the empirical observations

exactly, but it may not be able to make accurate predictions about unseen instances; *e.g.*, consider constructing a lookup table from observed data points to their responses. This phenomenon is known as *overfitting* to the training data. One possibility to avoid overfitting is to only consider a small or restricted space of hypotheses; *e.g.*, the space linear functions. Alternatively, one could allow for a large space of hypotheses, but penalize the complexity of a hypothesis—a practice known as *regularization*. Thus, the learner selects a hypothesis $f^\dagger$ that minimizes the modified objective

$$\tilde{R}_N(f) + \lambda \cdot \rho(f) \tag{2.2}$$

where the function $\rho : \mathcal{F} \to \Re$ is a measure of the complexity of a hypothesis and $\lambda \in \Re^+$ is a parameter that controls the trade-off between risk minimization and hypothesis complexity.

Prediction/Evaluation: Once trained on a dataset, the learned hypothesis is subsequently used to predict the response variables for a set of unlabeled data. I call this the evaluation phase although it may also be referred to as the test or prediction phase. Initially, only the data point x is available to the predictor. The learned hypothesis $f^\dagger$ predicts a value $\hat{y} = f^\dagger(x)$ in the space $\mathcal{Y}$ of all possible responses[2]. Finally, the actual label y is revealed and the agent receives a loss $L(\hat{y}, y)$ as an assessment of its performance. In the classification setting, there are generally two types of classification mistakes: a *false positive* (FP) is a normal instance classified as positive and a *false negative* (FN) is a malicious instance classified as negative. Selecting an appropriate trade-off between false positives and false negatives is an application-specific task.

The performance of a learner is typically assessed on a held-out set of labeled evaluation dataset, $\mathbb{D}^{(\text{eval})}$. Predictions are generated by $f^\dagger$ for each data point $x^{(i)} \in \mathbb{D}_X^{(eval)}$ in the evaluation dataset and the losses incurred are aggregated into various performance measures. In the classification setting, the typically performance measures are the *false positive rate* (FPR), the fraction of negative instances classified as positive, and the *false negative rate* (FNR), the fraction of positive instances classified as negatives. Often a classifier is tuned to have a particular (empirical) false positive rate based on held-out training data (validation dataset) and its resulting false negative rate is assessed at that FP-level.

2.2.5 Other Learning Paradigms

It is also interesting to consider cases where a classifier has more than two classes, or even a real-valued output. Indeed, the spam filter SpamBayes, which I study in Chapter 4, uses a third label, *unsure*, to allow the end-user to examine these potential spam more closely. However, generalizing the analysis of errors to more than two classes is not straightforward, and furthermore most systems in practice make a single fundamental distinction (for example, regardless of the label applied by the spam filter, the end-user will ultimately decide to treat each class as either junk messages or legitimate mail). For these reasons, and in keeping with common practice in the literature, I limit my analysis to binary classification and leave extensions to the multi-class or real-valued prediction as future work.

[2]The space of allowed predictions or actions $\mathcal{A}$ need not be the same as the space of allowed responses, $\mathcal{Y}$. This allows the learner to choose from a larger range of responses (hedging bets) or to restrict the learner to some desired subset. However, unless explicitly stated, I will assume $\mathcal{A} = \mathcal{Y}$.

In Chapter 5, I also study an anomaly detection setting. Like binary classification, anomaly detection consists of making one of two predictions: the data is normal ('$-$') or the data is anomalous ('+'). Unlike the classification setting, training data usual only consists of examples from the negative class. Because of this, it is common practice to calibrate the detector to achieve a desired false positive rate on held-out training data.

There are other interesting learning paradigms to consider such as semi-supervised, unsupervised, and reinforcement learning. However, as they do not directly impact my dissertation, I will not discuss these frameworks. For a thorough discussion of different learning settings refer to Hastie et al. [2003] or Mitchell [1997].

Chapter 3

A Framework for Secure Learning

The study of learning in adversarial environments is a relatively new discipline at the intersection between machine learning and computer security. I introduce a framework for qualitatively assessing the security of a machine learning system that captures a broad set of security characteristics common to a number of related adversarial learning settings. There has been a rich set of work in recent years that examines the security of machine learning systems, and here, I survey prior studies of learning in adversarial environments, attacks against learning systems and proposals for making systems secure against attacks. I identify different classes of attacks on machine learning systems (Section 3.3) and organize these attacks in terms of a taxonomy and a secure learning game, demonstrating that this framework captures the salient aspects of each attack.

While many researchers have considered particular attacks on machine learning systems, this chapter presents a comprehensive view of attacks. I organize attacks against machine learning systems based on a taxonomy that categorizes a threat in terms of three crucial properties of such attacks. I also present secure learning as a game between an *attacker* and a *defender*; the taxonomy determines the structure of the game and its cost model. Further, this taxonomy provides a basis for evaluating the resilience of the systems described by analyzing threats against them to construct defenses. The development of defensive learning techniques is more tentative, but I also discuss a variety of techniques that show promise for defending against different types of attacks.

The work I present not only provides a common language for thinking and writing about secure learning, but goes beyond that to show how the framework applies to both algorithm design and the evaluation of real-world systems. Not only does the framework elicit common themes in otherwise disparate domains, it has also motivated my study of practical machine learning systems as presented in Chapters 4, 5, and 6. These foundational principals for characterizing attacks against learning systems are an essential first step if secure machine learning is to reach its potential as a tool for use in real systems in security-sensitive domains.

This work was first introduced in the paper *Can Machine Learning be Secure?* [Barreno et al., 2006] that I wrote with my co-workers for *ASIACCS'06*. This work was later expanded and used to categorize prior work in the secure learning field in our paper *The Security of Machine Learning* published in *Machine Learning* [Barreno et al., 2010] and in Marco Barreno's dissertation [Barreno, 2008]. Here I use this framework as the central organizing scheme for my dissertation, my methodology, and the prior work in this field.

3.1 Analyzing Phases of Learning

Attacks can occur at each of the phases of the learning process that were outlined in Chapter 2.2. Figure 2.1(a) depicts how data flows through each phase of learning. I briefly outline how attacks against these phases differ.

The Measuring Phase With knowledge of the measurement process, an adversary can design malicious instances to mimic the measurements of innocuous data. After a successful attack against the measurement mechanism, the system may require expensive re-instrumentation or redesign to accomplish its task.

The Feature Selection Phase The feature selection process can be attacked in the same manner as the measuring phase except countermeasures and recovery are less costly since feature selection is a dynamic process that can be more readily adapted. Potentially, re-training could even be automated. However, feature selection can also be attacked in the same manner as the training phase (below) if feature selection is based on training data that may be contaminated.

Learning Model Selection Once the learning model is known, an adversary could exploit assumptions inherent in the model. Erroneous or unreasonable modeling assumptions about the training data may be exploited by an adversary; *e.g.*, if a model erroneously assumes linear separability in the data, the adversary could use data that can not be separated linearly to deceive the learner or make it perform poorly. It is essential to explicitly state and critique the modeling assumptions to identify potential vulnerabilities since changing the model may require that the system be redesigned.

The Training Phase By understanding how the learner trains, an adversary can design data to fool the learner into choosing a poor hypothesis. Robust learning methods are promising techniques to counter these attacks as discussed in Section 3.5.4.3. These methods are resilient to adversarial contamination although there are inherent trade-offs between their robustness and performance.

The Prediction Phase Once learned, an imperfect hypothesis can be exploited by an adversary who discovers prediction errors made by the learner. Assessing how difficult it is to discover such errors is an interesting question; *e.g.*, the ACRE-learning framework of Lowd and Meek [2005b] as discussed further in Chapter 3.4.4. An interesting avenue of future research is detecting that an adversary is exploiting these errors and retraining to counter the attack.

To better understand these different attacks, consider a spam filter: that (*i*) has some simple set of measurements of email such as *hasAttachment*, *subjectLength*, *bodyLength*, *etc.*, (*ii*) selects the top-ten most frequently appearing features in spam, (*iii*) uses the naive Bayes model, (*iv*) trains class frequencies by empirical counts, and (*v*) classifies email by thresholding the model's predicted class probabilities. An attack against the measurement

(or feature selection) phase would consist of determining the features used (for classification) and producing spams that are indistinguishable from normal email for those features. An attack against the learning model would entail discovering a set of spams and hams that could not be classified correctly due to the linearity of the naive Bayes boundary. Further, the training system (or feature selection) could be attacked by injecting spams with misleading spurious features causing it to learn the wrong hypothesis. Finally, the prediction phase could be attacked by systematically probing the filter to find spams that are misclassified as ham (false negatives).

Many learning methods make a stationarity assumption: training data and evaluation data are drawn from the same distribution. Under this assumption minimizing the risk on the training set is a surrogate for risk on the evaluation data. However, real-world sources of data often are not stationary and, even worse, attackers can easily break the stationarity assumption with some control of either training or evaluation instances. Analyzing and strengthening learning methods to withstand or mitigate violations of the stationarity assumption is the crux of the *secure learning* problem.

Qualifying the vulnerable components of the learning system is only the first step to understanding the adversary. In the next section, I outline a framework my colleagues and I designed to qualify the adversary's goals.

3.2 Security Analysis

Security is concerned with protecting assets from attackers. Properly analyzing the security of a system requires identifying the security goals and a threat model for the system. A *security goal* is a requirement that, if violated, results in the partial or total compromise of an asset. A threat model is a profile of attacker who wish to harm the system, describing their motivation and capabilities. Here I describe the security goals and threat model for machine learning systems.

In a security-sensitive domain, classifiers can be used to make distinctions that advance the security goals of the system. For example, a *virus detection system* has the goal of reducing susceptibility to virus infection, either by detecting the virus in transit prior to infection or by detecting an extant infection to expunge. Another example is an *intrusion detection system* (IDS), which has the goal of preventing harm from malicious intrusion, either by identifying existing intrusions for removal or by detecting malicious traffic and preventing it from reaching its intended target[1]. In this section, I describe security goals and a threat model that are specific to machine learning systems.

3.2.1 Security Goals

In a security context the classifier's purpose is to classify malicious events and prevent them from interfering with system operations. We split this general learning goal into two goals:

- **Integrity goal**: To prevent attackers from reaching system assets.

[1]In the case of preventing intrusion, the whole system is more properly called an *intrusion prevention system* (IPS). I have no need to distinguish between the two cases, so I use IDS to refer to both intrusion detection systems and intrusion prevention systems.

- **Availability goal**: To prevent attackers from interfering with normal operation.

There is a clear connection between false negatives and violation of the integrity goal: malicious instances that pass through the classifier can wreak havoc. Likewise, false positives tend to violate the availability goal because the learner itself denies benign instances.

3.2.2 Threat Model

Attacker goal and incentives. In general the attacker wants to access system assets (typically with false negatives) or deny normal operation (usually with false positives). For example, a virus author wants viruses to pass through the filter and take control of the protected system (a false negative). On the other hand, an unscrupulous merchant may want sales traffic to a competitor's web store to be blocked as intrusions (false positives).

We assume that the attacker and defender each have a *cost function* that assigns a cost to each labeling for any given instance. Cost can be positive or negative; a negative cost is a benefit. It is usually the case that low cost for the attacker parallels high cost for the defender and vice-versa; the attacker and defender would not be adversaries if their goals aligned. Unless otherwise stated, for ease of exposition I assume that every cost for the defender corresponds to a similar benefit for the attacker and vice-versa. This assumption is not essential to this framework, which extends easily to arbitrary cost functions, but not necessary for my exposition. In this chapter, I take the defender's point of view and use "high-cost" to mean high positive cost for the defender.

3.2.2.1 Attacker capabilities

I assume that the attacker has knowledge of the training algorithm, and in many cases partial or complete information about the training set, such as its distribution. For example, the attacker may have the ability to eavesdrop on all network traffic over the period of time in which the learner gathers training data. I examine different degrees of the attacker's knowledge and assess how much he gains from different sources of potential information.

In general, I assume the attacker can generate arbitrary instances; however, many settings do impose significant restrictions on the instances generated by the attacker. For example, when the learner trains on data from the attacker, sometimes it is safe to assume that the attacker cannot choose the label for training, such as when training data is carefully hand labeled. As another example, an attacker may have complete control over data packets being sent from the attack source, but routers in transit may add to or alter the packets as well as affect their timing and arrival order.

I assume the attacker has the ablity to modify or generate data used in training and explore scenarios both when he has this capability and when he does not. When the attacker controls training data, an important limitation to consider is what fraction of the training data the attacker can control and to what extent. If the attacker has arbitrary control over 100% of the training data, it is difficult to see how the learner can learn anything useful; however, even in such cases there are learning strategies that can make the attacker's task more difficult (see Section 3.6). I examine intermediate cases and explore how much influence is required for the attacker to defeat the learning procedure.

	Integrity	*Availability*
Causative: *Targeted*	Kearns and Li [1993], Newsome et al. [2006]	Kearns and Li [1993], Newsome et al. [2006], Chung and Mok [2007], Nelson et al. [2008]
Indiscriminate	Kearns and Li [1993], Newsome et al. [2006]	Kearns and Li [1993], Newsome et al. [2006], Chung and Mok [2007], Nelson et al. [2008]
Exploratory: *Targeted*	Tan et al. [2002], Lowd and Meek [2005a], Wittel and Wu [2004], Lowd and Meek [2005b]	Moore et al. [2006]
Indiscriminate	Fogla and Lee [2006], Lowd and Meek [2005a], Wittel and Wu [2004]	Moore et al. [2006]

Table 3.1: Related work in the taxonomy.

3.3 Framework

The framework I describe here has three primary components: a taxonomy based on the common characteristics of attacks against learning algorithms, a high-level description of the elements of the game played between the attacker and defender (learner), and set of common characteristics for an attacker's capabilities. Each of these elements help organize and assess the threat posed by an attacker.

3.3.1 Taxonomy

A great deal of the work that has been done within secure learning is the analysis of attack and defense scenarios for particular learning applications. My colleagues and I developed a qualitative taxonomy of attacks against machine learning systems which we used both to categorize others research, to find commonalities between otherwise disparate domains, and ultimately to frame our own research. Here, I present a taxonomy categorizing attacks against learning systems along three axes. Each of these dimensions operates independently, so we have at least eight distinct classes of attacks on machine learning system. This taxonomy divides threats as follows:

Influence

- *Causative* attacks influence learning with control over training data.
- *Exploratory* attacks exploit misclassifications but do not affect training.

Security violation

- *Integrity* attacks compromise assets via false negatives.
- *Availability* attacks cause denial of service, usually via false positives.

Specificity

- *Targeted* attacks focus on a particular instance.
- *Indiscriminate* attacks encompass a wide class of instances.

The first axis describes the capability of the attacker: whether (a) the attacker has the ability to influence the training data that is used to construct the classifier (a *Causative* attack) or (b) the attacker does not influence the learned classifier, but can send new instances to the classifier and possibly observe its decisions on these carefully crafted instances (an *Exploratory* attack).

The second axis indicates the type of security violation the attacker causes: either (a) allowing harmful instances to slip through the filter as false negatives (an *Integrity* violation); or (b) creating a denial of service event in which benign instances are incorrectly filtered as false positives (an *Availability* violation).

The third axis refers to how specific the attacker's intention is: whether (a) the attack is highly *Targeted* to degrade the classifier's performance on one particular instance or (b) the attack aims to cause the classifier to fail in an *Indiscriminate* fashion on a broad class of instances. Each axis, especially this one, can actually be a spectrum of choices, but for simplicity, I will categorize attacks and defenses into these groupings.

These axes define the space of attacks against learners and aid in identifying unconventional threats. By qualifying where an attack lies in this space, one can begin to quantify the adversary's capabilities and assess the risk posed by this threat. Laskov and Kloft [2009] have since extended these basic principles to propose a framework for quantitatively evaluating security threats.

3.3.2 The Adversarial Learning Game

I model the task of constructing a secure learning system as a game between an attacker and a defender—the attacker manipulates data to mis-train or evade a learning algorithm chosen by the defender to thwart the attacker's objective. The characteristics specified by the taxonomy's axes also designate some aspects of this game. The INFLUENCE axis determines the structure of the game and the legal moves that each player can make. The SPECIFICITY and SECURITY VIOLATION axes of the taxonomy determine the general shape of the cost function: an *Integrity* attack benefits the attacker on false negatives, and therefore focuses high cost (to the defender) on false negatives, and an *Availability* attack focuses high cost on false positives; a *Targeted* attack focuses high cost only on a small number of instances, while an *Indiscriminate* attack spreads high cost over a broad range of instances.

I formalize the game as a series of *moves*, or *steps*. Each move either is a strategic choice by one of the players or is a *neutral* move not controlled by either player. The choices and

computations in a move depend on information produced by previous moves (when a game is repeated, this includes previous iterations) and on domain-dependent constraints which I highlight in discussing prior work. Generally, though, in an *Exploratory* attack, the attacker chooses a procedure $A^{(\text{eval})}$ that affects the evaluation data $\mathbb{D}^{(\text{eval})}$, and in a *Causative* attack, the attacker also chooses a procedure $A^{(\text{train})}$ to manipulate the training data $\mathbb{D}^{(\text{train})}$. In either setting, the defender chooses a learning algorithm $H^{(N)}$. This formulation gives us a theoretical basis for analyzing the interactions between attacker and defender.

3.3.3 Characteristics of Adversarial Capabilities

In this section I introduce three essential properties for constructing a model of an attack against a learning algorithm that refine the game played between the learner and the adversary as described by the taxonomy. These properties define a set of common domain-specific adversarial limitations that allow a security analyst to formally describe the capabilities of the adversary.

3.3.3.1 Corruption Models

The most important aspect of the adversary is how he can alter data to mislead or evade the classifier. As previously stated, learning against an unlimited adversary is futile. Instead, the security analysis I propose focuses on a limited adversary, but to do so, one must model the restrictions on the adversary and justify these restrictions for a particular domain. Here, I outline two common models for adversarial corruption, and I describe how the adversary is limited within each.

Data Insertion Model: The first model assumes the adversary has unlimited control of a small fraction of the data; *i.e.*, the adversary is restricted to only modify a limited amount of data but can alter those data points arbitrarily. I call this an *insertion model* because, in this scenario, the adversary crafts a small number of attack instances and *inserts* them into the dataset for training or evaluation (or perhaps replaces existing data points). For example, in the example of a spam filter, the adversary (spammer) can create any arbitrary message for their attack but he is limited in the number of attack messages he can inject; thus, the spammer's attack on the spam filter can be analyzed in terms of how many messages are required for the attack to be effective. For this reason, I use this model of corruption in analyzing attacks on the SpamBayes spam filter in Chapter 4 and show that even with a relatively small number of attack messages, the adversarial spammer can significantly mislead the filter.

Data Alteration Model: The second corruption model instead assumes that the adversary can alter any (or all) of the data points in the data set but is limited in the degree of alteration; *i.e.*, a *alteration model*. For example, to attack a detector that is monitoring network traffic volumes over windows of time, the adversary can add or remove traffic within the network but only can make a limited degree of alteration. Such an adversary cannot insert new data since each data point corresponds to a time slice and the adversary cannot arbitrarily control any single data point since other actors are also creating traffic in

the network. Here, the adversary is restricted by the total amount of alteration they make, and so the effectiveness of his attack can be analyzed in terms of the size of alteration required to achieve the attacker's objective. This is the model I use for analyzing attacks on a PCA-subspace detector for network anomaly detection in Chapter 5 and again I show that with a relatively small degree of control, the adversary can dramatically degrade the effectiveness of this detector using data alterations.

3.3.3.2 Class Limitations

A second limitation on attackers involves which parts of the data the adversary is allowed to alter—the positive (malicious) class, the negative (benign) class, or both. Usually, attackers external to the system are only able to create malicious data and so they are limited to only manipulating positive instances. This is the model I use throughout dissertation. However, there is also an alternative threat that *insiders* could attack a learning system by altering negative instances. I do not analyze this threat in this thesis but return to the issue in the discussion in Chapter 7.

3.3.3.3 Feature Limitations

The final type of adversarial limitation I consider are limits on how an adversary can alter data points in terms of each *feature*. Features represent different aspects of the state of the world and have various degrees of vulnerability to attack. Some features can be arbitrarily changed by the adversary, but others may have stochastic aspects that the adversary cannot completely control, and some features may not be alterable at all. For instance, in sending an email, the adversary can completely control the content of the message but cannot completely determine the routing of the message or its arrival time. Further, this adversary has no control over meta-information that is added to the message by mail relays while the message is en route. Providing an accurate description of the adversary's control over the features is essential.

3.3.4 Attacks

In the remainder of this chapter, I survey prior research, I discuss how attack and defense strategies were developed in different domains, I reveal their common themes, and I highlight important aspects of the secure learning game in the context of this taxonomy. The related work discussed below is also presented in the taxonomy in Table 3.1. For an *Exploratory* attack, I discuss realistic instances of the attacker's choice for $A^{(\text{eval})}$ in Sections 3.4.2 and 3.4.3. Similarly, in Sections 3.5.2 and 3.5.3, I discuss practical examples of the attacker's choices in the *Causative* game. Finally, in Section 3.7, I organize the remainder of my dissertation within the context of this framework.

3.3.5 Defenses

The game between attacker and defender and the taxonomy also provide a foundation on which to construct defense strategies against broad classes of attacks. I address *Exploratory*

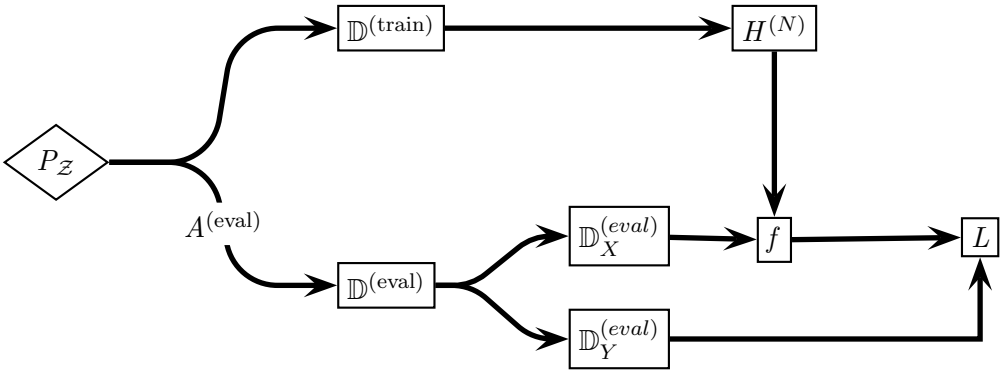


Figure 3.1: Diagram of an Exploratory attack against a learning system (see Figure 2.1).

and *Causative* attacks separately. For *Exploratory* attacks, I discuss the defender's choice for an algorithm $H^{(N)}$ in Section 3.4.4 and I discuss the defender's strategies in a *Causative* setting in Section 3.5.4. Finally, in Section 3.6, I discuss the broader setting of an iterated game.

In all cases, defenses present a trade-off: changing the algorithms to make them more robust against (worst-case) attacks will generally make them *less* effective on non-adversarial data. Analyzing this trade-off is an important part of developing defenses.

3.4 Exploratory Attacks

Based on the Influence axis of the taxonomy, the first category of attacks that I discuss are *Exploratory* attacks, which influence only the evaluation data as indicated in Figure 3.1. The adversary's transformation $A^{(\text{eval})}$ alters the evaluation data either by defining a procedure to change instances drawn from $P_{\mathcal{Z}}$ or by changing $P_{\mathcal{Z}}$ to an altogether different distribution $P_{\mathcal{Z}}^{(\text{eval})}$ chosen by the adversary. The adversary makes these changes based on (partial) information gleaned about the training data $\mathbb{D}^{(\text{train})}$, the learning algorithm $H^{(N)}$, and the classifier f. Further, the adversary's transformation may evolve as the adversary learns more about the classifier with each additional prediction it makes.

3.4.1 The Exploratory Game

First I present the formal version of the game for *Exploratory* attacks, and then explain it in greater detail.

1. **Defender** Choose procedure $H^{(N)}$ for selecting hypothesis
2. **Attacker** Choose procedure $A^{(\text{eval})}$ for selecting an evaluation distribution
3. Evaluation:
   - Reveal distribution $P_{\mathcal{Z}}^{(\text{train})}$
   - Sample dataset $\mathbb{D}^{(\text{train})}$ from $P_{\mathcal{Z}}^{(\text{train})}$
   - Compute $f \leftarrow H^{(N)}\left(\mathbb{D}^{(\text{train})}\right)$
   - Compute $P_{\mathcal{Z}}^{(\text{eval})} \leftarrow A^{(\text{eval})}(\mathbb{D}^{(\text{train})}, f)$
   - Sample dataset $\mathbb{D}^{(\text{eval})}$ from $P_{\mathcal{Z}}^{(\text{eval})}$
   - Assess total cost: $\sum_{\langle x,y\rangle \in \mathbb{D}^{(\text{eval})}} L_x\left(f(x), y\right)$

The defender's move is to choose a learning algorithm (procedure) $H^{(N)}$ for creating hypotheses from datasets. Many procedures used in machine learning have the form of Equation (2.2). For example, the defender may choose a *support vector machine* (SVM) with a particular kernel, loss, regularization, and cross-validation plan. The attacker's move is then to choose a procedure $A^{(\text{eval})}$ to produce a distribution on which to evaluate the hypothesis that $H^{(N)}$ generates. (The degree of control the attacker has in generating the dataset and the degree of information about $\mathbb{D}^{(\text{train})}$ and f that $A^{(\text{eval})}$ has access to are setting-specific.)

After the defender and attacker have both made their choices, the game is evaluated. A training dataset $\mathbb{D}^{(\text{train})}$ is drawn from some fixed and possibly unknown distribution $P_{\mathcal{Z}}^{(\text{train})}$, and training produces $f = H^{(N)}\left(\mathbb{D}^{(\text{train})}\right)$. The attacker's procedure $A^{(\text{eval})}$ produces distribution $P_{\mathcal{Z}}^{(\text{eval})}$, which is based in general on $\mathbb{D}^{(\text{train})}$ and f, and an evaluation dataset $\mathbb{D}^{(\text{eval})}$ is drawn from $P_{\mathcal{Z}}^{(\text{eval})}$. Finally, the attacker and defender incur cost based on the performance of f evaluated on $\mathbb{D}^{(\text{eval})}$ according to the loss function $L_x(\cdot, \cdot)$. Note that, unlike in Chapter 2.2, here I allow the loss function to depend on the data point x. This generalization allows this game to account for an adversary (or learner) with instance-dependent costs [*cf.*, Dalvi et al., 2004].

The procedure $A^{(\text{eval})}$ generally depends on $\mathbb{D}^{(\text{train})}$ and f, but the amount of information an attacker actually has is setting specific (in the least restrictive case the attacker knows $\mathbb{D}^{(\text{train})}$ and f completely). The attacker may know a subset of $\mathbb{D}^{(\text{train})}$ or the family $\mathcal{F}$ of f. However, the procedure $A^{(\text{eval})}$ may also involve acquiring information dynamically. For instance, in many cases, the procedure $A^{(\text{eval})}$ can *query* the classifier, treating it as an oracle that provides labels for query instances; this is one particular degree of information that $A^{(\text{eval})}$ can have about f. Attacks that use this technique are *probing attack*. Probing can reveal information about the classifier. On the other hand, with sufficient prior knowledge about the training data and algorithm, the attacker may be able to find high-cost instances without probing.

3.4.2 Exploratory Integrity Attacks

The most frequently studied attacks are *Exploratory Integrity* attacks in which the adversary attempts to passively circumvent the learning mechanism to exploit blind spots in the learner that allow miscreant activities to go undetected. In an *Exploratory Integrity* attack, the attacker crafts intrusions so as to evade the classifier without direct influence over the classifier itself. Instead, attacks of this sort often attempt to systematically make the miscreant activity appear to be normal activity to the detector or obscure the miscreant activity's identifying characteristics. Some *Exploratory Integrity* attacks mimic statistical properties of the normal traffic to camouflage intrusions; *e.g.*, the attacker examines training data and the classifier, then crafts intrusion data. In the *Exploratory* game, the attacker's move produces malicious instances in $\mathbb{D}^{(\text{eval})}$ that statistically resemble normal traffic in the training data $\mathbb{D}^{(\text{train})}$.

Example 3.1 (The Shifty Intruder)
An attacker modifies and obfuscates intrusions, such as by changing network headers and reordering or encrypting contents. If successful, these modifications prevent the IDS from recognizing the altered intrusions as malicious, so it allows them into the system. In the *Targeted* version of this attack, the attacker has a particular intrusion to get past the filter. In the *Indiscriminate* version, the attacker has no particular preference and can search for any intrusion that succeeds, such as by modifying a large number of different exploits to see which modifications evade the filter.

3.4.2.1 Polymorphic blending attack

Fogla and Lee [2006] introduce *polymorphic blending attacks* that evade intrusion detectors using encryption techniques to make attacks statistically indistinguishable from normal traffic. They present a formalism for reasoning about and generating *polymorphic blending attack* instances to evade intrusion detection systems. The technique is fairly general and is *Indiscriminate* in which intrusion packets it modifies.

Feature deletion attacks instead specifically exclude high-value identifying features used by the detector [Globerson and Roweis, 2006]; this form of attack stresses the importance of proper feature selection as was also demonstrated empirically by Mahoney and Chan [2003] in their study of the behavior of intrusion detection systems on the DARPA/Lincoln Lab dataset.

3.4.2.2 Attacking a sequence-based IDS

Tan et al. [2002] describe a *mimicry* attack against the `stide` sequence-based intrusion detection system (IDS) proposed by Forrest et al. [1996], Warrender et al. [1999]. They modify exploits of the `passwd` and `traceroute` programs to accomplish the same ends using different sequences of system calls: the shortest subsequence in attack traffic that does not appear in normal traffic is longer than the IDS window size. By exploiting the finite window size of the detector, this technique makes attack traffic indistinguishable from normal traffic for the detector. This attack is more *Targeted* than polymorphic blending since it modifies particular intrusions to look like normal traffic. In subsequent work Tan et al. [2003] characterize their attacks as part of a larger class of information hiding techniques

which they demonstrate can make exploits mimic either normal call sequences or the call sequence of another less severe exploit.

Independently, Wagner and Soto [2002] have also developed mimicry attacks against a sequence-based IDS called pH proposed by Somayaji and Forrest [2000]. Using the machinery of finite automata, they construct a framework for testing whether an IDS is susceptible to mimicry for a particular exploit. In doing so, they develop a tool for validating IDSs on a wide-range of variants of a particular attack and suggest that similar tools should be more broadly employed to identify the vulnerabilities of an IDS.

Overall, these mimicry attacks against sequence-based anomaly detection systems underscore critical weaknesses in these systems that allow attackers to obfuscate the necessary elements of their exploits to avoid detection by mimicking normal behaviors. Further they highlight how an IDS may appear to perform well against a known exploit but, unless it captures necessary elements of the intrusion, the exploit can easily be adapted to circumvent the detector. See Section 3.4.4 for more discussion.

3.4.2.3 Good word attacks

Adding or changing words in a spam message can allow it to bypass the filter. Like the attacks against an IDS above, these attacks all use both training data and information about the classifier to generate instances intended to bypass the filter. They are somewhat independent of the *Targeted/Indiscriminate* distinction, but the *Exploratory* game captures the process used by all of these attacks.

Studying these techniques was first suggested by John Graham-Cumming. In a presentation *How to Beat an Adaptive Spam Filter* at the 2004 MIT Spam Conference, he presented a *Bayes vs. Bayes* attack that uses a second statistical spam filter to find good words based on feedback from the filter under attack. Several authors have further explored evasion techniques used by spammers and demonstrated attacks against spam filters using similar principles as those against IDSs as discussed above. Lowd and Meek [2005a] and Wittel and Wu [2004] develop attacks against statistical spam filters that add *good words*, or words the filter considers indicative of non-spam, to spam emails. This *good word attack* makes spam emails appear innocuous to the filter, especially if the words are chosen to be ones that appear often in non-spam email and rarely in spam email. Finally, obfuscation of spam words (*i.e.*, changing characters in the word or the spelling of the word so it no longer recognized by the filter) is another popular technique for evading spam filters which has been formalized by several authors (*cf.* Liu and Stamm [2007] and Sculley et al. [2006]).

3.4.2.4 Cost-based Evasion

Another vein of research focuses on the costs incurred due to the adversary's evasive actions; *i.e.*, instances that evade detection may be less desirable to the adversary. In using costs, this work explicitly casts evasion as a problem where the adversary wants to evade detection but wants to do so using high-value instances (an assumption that was implicit in the other work discussed in this section). Dalvi et al. [2004] exploit these costs to develop a cost-sensitive game-theoretic classification defense that is able to successfully detect optimal evasion of the original classifier. Using this game-theoretic approach, this technique preemptively patches

the naive classifier's blind spots by constructing a modified classifier designed to detect optimally modified instances.

Subsequent game theoretic approaches to learning have extended this setting and solved for an equilibrium for the game [Brückner and Scheffer, 2009, Kantarcioglu et al., 2009]. Further, Biggio et al. [2010] extend this game theoretic approach and propose hiding information or randomization as additional defense mechanisms for this setting.

Cost models of the adversary also led to a theory for query-based near-optimal evasion of classifiers first presented by Lowd and Meek [2005b] in which they cast the difficulty of evading a classifier into a complexity problem. They give algorithms for an attacker to reverse engineer a classifier. The attacker seeks the highest cost (lowest cost for the attacker) instance that the classifier labels *negative*. In *Near-Optimal Evasion of Convex-Inducing Classifiers*, I published an extension to this work with my colleagues [Nelson et al., 2010a]. I generalized the theory of near-optimal evasion to a broader class of classifiers and demonstrated that the problem is easier than reverse-engineering approaches; work that I thoroughly explain in Chapter 6.

3.4.3 Exploratory Availability Attacks

In an *Exploratory Availability* attack, the attacker interferes with the normal behavior of a learning system without influence over training. This type of attack against non-learning systems abound in the literature: almost any denial-of-service (DoS) attack falls into this category, such as those described by Moore et al. [2006]. However, *Exploratory Availability* attacks against the learning components of systems are not common and I am not aware of any studies of them. It seems the motivation for attacks of this variety is not as compelling as other attacks against learners.

One possible attack is described in the example below: if a learning IDS has trained on intrusion traffic and has the policy of blocking hosts that originate intrusions, an attacker could send intrusions that appear to originate from a legitimate host, convincing the IDS to block that host. Another possibility is to take advantage of a computationally expensive learning component: for example, spam filters that use image processing to detect advertisements in graphical attachments can take significantly more time than text-based filtering [Dredze et al., 2007, Wang et al., 2007]. An attacker could exploit such overhead by sending many emails with images, causing the expensive processing to delay and perhaps even block messages.

Example 3.2 (The Mistaken Identity)
An attacker sends intrusions that appear to come from the IP address of a legitimate machine. The IDS, which has learned to recognize intrusions, blocks that machine. In the *Targeted* version, the attacker has a particular machine to target. In the *Indiscriminate* version, the attacker may select any convenient machine or may switch IP addresses among many machines to induce greater disruption.

3.4.4 Defending against Exploratory Attacks

Exploratory attacks do not corrupt the training data but attempt to find vulnerabilities in the learned hypothesis. Through control over the evaluation data, the attacker can violate

the assumption of stationarity. When producing the evaluation distribution, the attacker attempts to construct an *unfavorable evaluation distribution* concentrating probability mass on high-cost instances; in other words, the attacker's procedure $A^{(\text{eval})}$ constructs an evaluation distribution $P_{\mathcal{Z}}^{(\text{eval})}$ on which the learner predicts poorly (thus violating stationarity); *i.e.*, the attacker chooses $P_{\mathcal{Z}}^{(\text{eval})}$ to maximize the cost computed in the last step of the *Exploratory* game. This section examines defender strategies that make it difficult for the attacker to construct such a distribution.

In the *Exploratory* game, the defender makes a move before observing contaminated data; that is, here I do not consider scenarios where the defender is permitted to react to the attack. The defender can impede the attacker's ability to reverse engineer the classifier by limiting access to information about the training procedure and data. With less information, $A^{(\text{eval})}$ has difficulty producing an unfavorable evaluation distribution. Nonetheless, even with incomplete information, the attacker may be able to construct an unfavorable evaluation distribution using a combination of *prior knowledge* and *probing*.

The defender's task is to design data collection and learning techniques that make it difficult for an attacker to reverse engineer the hypothesis. The primary task in analyzing *Exploratory* attacks is quantifying the attacker's ability to reverse engineer the learner.

3.4.4.1 Defenses against attacks without probing

Part of a security analysis involves identifying aspects of the system that should be kept secret. In securing a learner, the defender can limit information to make it difficult for an attacker to conduct their attack.

Training data: Preventing the attacker from knowing the training data limits the attacker's ability to reconstruct internal states of the classifier. There is a tension between collecting training data that fairly represents the real world instances and keeping all aspects of that data secret. In most situations, it is difficult to use completely secret training data, though the attacker may have only partial information about it.

Feature selection: The defender can also harden classifiers against attacks through attention to features in the feature selection and learning steps (which are both internal steps of the defender's hypothesis selection procedure $H^{(N)}$). Feature selection is the process of choosing a feature map that transforms raw measurements into the feature space used by the learning algorithm. In the learning step, the learning algorithm builds its model or signature using particular features from the map's feature space; this choice of features for the model or signature is also sometimes referred to as feature selection, though I consider it to be part of the learning process, after the feature map has been established. For example, one feature map for email message bodies might transform each token to a Boolean feature indicating its presence; another map might specify a real-valued feature indicating the relative frequency of each word in the message compared to its frequency in natural language; yet another map might count sequences of n characters and specify an integer feature for each character n-gram indicating how many times it appears. In each of these cases, a learner will construct a model or signature that uses certain features (tokens present

or absent; relative frequency of words present; character n-gram counts) to decide whether an instance is benign or malicious.

Obfuscation of spam-indicating words (an attack on the feature set) is a common *Targeted Exploratory Integrity* attack. Sculley et al. [2006] use inexact string matching in feature selection to defeat obfuscations of words in spam emails. They choose a feature map based on character subsequences that are robust to character addition, deletion, and substitution.

Globerson and Roweis [2006] present a feature-based learning defense for the *feature deletion attack*; an *Exploratory* attack on the evaluation data $\mathbb{D}^{(\text{eval})}$. In feature deletion, features present in the training data, and perhaps highly predictive of an instance's class, are removed from the evaluation data by the attacker. For example, words present in training emails may not occur in evaluation messages, and network packets in training data may contain values for optional fields that are missing from future traffic. Globerson and Roweis formulate a modified support vector machine classifier that is robust in its choice of features against deletion of high-value features.

One particularly important consideration when the learner builds its model or signature is to ensure that the learner uses features related to the intrusion itself. In their study of the DARPA/Lincoln Laboratory intrusion dataset, Mahoney and Chan [2003] demonstrate that spurious artifacts in training data can cause an IDS to learn to distinguish normal from intrusion traffic based on those artifacts rather than relevant features. Ensuring that the learner builds a model from features that describe the fundamental differences between malicious and benign instances should mitigate the effects of mimicry attacks (Section 3.4.2) and red herring attacks (Section 3.5.2).

Using spurious features in constructing a model or signature is especially problematic in cases where any given intrusion attempt may cause harm only probabilistically or depending on some internal state of the victim's system. If the features relevant to the intrusion are consistent for some set of instances but the actual cost of those instances varies widely, then a learner risks attributing the variation to other nonessential features.

Hypothesis space/learning procedures: A complex hypothesis space may make it difficult for the attacker to infer precise information about the learned hypothesis. However, hypothesis complexity must be balanced with capacity to generalize, such as through regularization.

Wang et al. [2006] present *Anagram*, an anomaly detection system using *n-gram* models of bytes to detect intrusions. They incorporate two techniques to defeat *Exploratory* attacks that mimic normal traffic (*mimicry attacks*): *i*) they use high-order n-grams (with n typically between 3 and 7), which capture differences in intrusion traffic even when that traffic has been crafted to mimic normal traffic on the single-byte level; and *ii*) they randomize feature selection by randomly choosing several (possibly overlapping) subsequences of bytes in the packet and testing them separately, so the attack will fail unless the attacker makes not only the whole packet but also any subsequence mimic normal traffic.

Dalvi et al. [2004] develop a cost-sensitive game-theoretic classification defense to counter *Exploratory Integrity* attacks. In their model, the attacker can alter natural instance features in $A^{(\text{eval})}$ but incurs a known cost for each change. The defender can

measure each feature at a different known cost. Each has a known cost function over classification/true label pairs. The classifier $H^{(N)}$ is a cost-sensitive naive Bayes learner that classifies instances to minimize his expected cost, while the attacker modifies features to minimize its own expected cost. Their defense constructs an adversary-aware classifier by altering the likelihood function of the learner to anticipate the attacker's changes. They adjust the likelihood that an instance is malicious by considering that the observed instance may be the result of an attacker's optimal transformation of another instance. This defense relies on two assumptions: *i*) the defender's strategy is a step ahead of the attacker's strategy (*i.e.*, their game differs from ours in that the attacker's procedure $A^{(\text{eval})}$ cannot take f into account), and *ii*) the attacker plays optimally against the original cost-sensitive classifier . It is worth noting that while their approach defends against optimal attacks, it doesn't account for non-optimal attacks. For example, if the attacker doesn't modify any data, the adversary-aware classifier misclassifies some instances that the original classifier correctly classifies.

3.4.4.2 Defenses against probing attacks

In the game described above in Section 3.4.1, the attacker selects an evaluation distribution $P_{\mathcal{Z}}^{(\text{eval})}$ for selecting the evaluation data $\mathbb{D}^{(\text{eval})}$ based on knowledge obtained from the training data $\mathbb{D}^{(\text{train})}$ and/or the classifier f. However, the procedure $A^{(\text{eval})}$ need not select a stationary distribution $P_{\mathcal{Z}}^{(\text{eval})}$. In fact, the attacker may incrementally change the distribution based on the observed behavior of the classifier to each data point generated from $P_{\mathcal{Z}}^{(\text{eval})}$—a *probing* or query-based adaptive attack. The ability for $A^{(\text{eval})}$ to query a classifier gives an attacker powerful additional attack options, which several researchers have explored.

Analysis of reverse engineering: Lowd and Meek [2005b] observe that the attacker need not model the classifier explicitly, but only find lowest-attacker-cost instances as in the setting of Dalvi et al. [2004]. They formalize a notion of reverse engineering as the adversarial classifier reverse engineering (ACRE) problem problem. Given an attacker cost function, they analyze the complexity of finding a lowest-attacker-cost instance that the classifier labels as negative. They assume no general knowledge of training data, though the attacker does know the feature space and also must have one positive example and one negative example. A classifier is *ACRE-learnable* if there exists a polynomial-query algorithm that finds a lowest-attacker-cost negative instance. They show that linear classifiers are ACRE-learnable with linear attacker cost functions and some other minor restrictions.

The ACRE-learning problem provides a means of qualifying how difficult it is to use queries to reverse engineer a classifier from a particular hypothesis class using a particular feature space. I now suggest defense techniques that can increase the difficulty of reverse engineering a learner.

Randomization: A randomized hypothesis may decrease the value of feedback to an attacker. Instead of choosing a hypothesis $f : \mathcal{X} \to \{0, 1\}$, I generalize to hypotheses that predict a real value on $[0, 1]$. This generalized hypothesis returns a probability of classifying $x \in \mathcal{X}$ as 1; *i.e.*, a *randomized* classifier. By randomizing, the expected performance of the

hypothesis may decrease on regular data drawn from a non-adversarial distribution, but it also may decrease the value of the queries for the attacker.

Randomization in this fashion does not reduce the information available in principle to the attacker, but merely requires more work from the attacker for the information. It is likely that this defense is appropriate in only a small number of scenarios.

Limiting/misleading feedback: Another potential defense is to limit the feedback given to an attacker. For example, common techniques in the spam domain include eliminating bounce emails, delivery notices, remote image loading, and other limits on potential feedback channels. In most settings, it is probably impossible to remove all feedback channels; however, limiting feedback increases work for the attacker. In some settings, it may also be possible to mislead the attacker by sending fraudulent feedback. Actively misleading the attacker by fabricating feedback suggests an interesting battle of information between attacker and defender. In some scenarios the defender may be able to give the attacker no information via feedback, and in others the defender may even be able to return feedback that causes the attacker to come to incorrect conclusions.

3.5 Causative Attacks

The second broad category of attacks from the taxonomy are *Causative* attacks, which influence the training data (as well as potentially subsequently modifying the evaluation data) as indicated in Figure 3.2. Again, the adversary's transformation $A^{(\text{eval})}$ alters the evaluation data either by defining a procedure to change instances drawn from $P_{\mathcal{Z}}$ or by changing $P_{\mathcal{Z}}$ to an alternative distribution $P_{\mathcal{Z}}^{(\text{eval})}$ chosen by the adversary (see Section 3.4). However, in addition to changing evaluation data, *Causative* attacks also allow the adversary to alter the training data with a second transformation $A^{(\text{train})}$, which either transforms instances drawn from $P_{\mathcal{Z}}$ or changes $P_{\mathcal{Z}}$ to an alternative distribution $P_{\mathcal{Z}}^{(\text{train})}$ during training. Of course, the adversary can synchronize $A^{(\text{train})}$ and $A^{(\text{eval})}$ to best achieve his desired objective, although in some *Causative* attacks, the adversary can only control the training data (*e.g.*, the attacker I describe in Chapter 4 can not control the non-spam messages sent during evaluation). Also note that, since the game described here is batch training, an adaptive procedure $A^{(\text{train})}$ is unnecessary although the distribution $P_{\mathcal{Z}}^{(\text{train})}$ can be non-stationary.

3.5.1 The Causative Game

The game for *Causative* attacks is similar to the game for *Exploratory* attacks with an augmented move for the attacker.

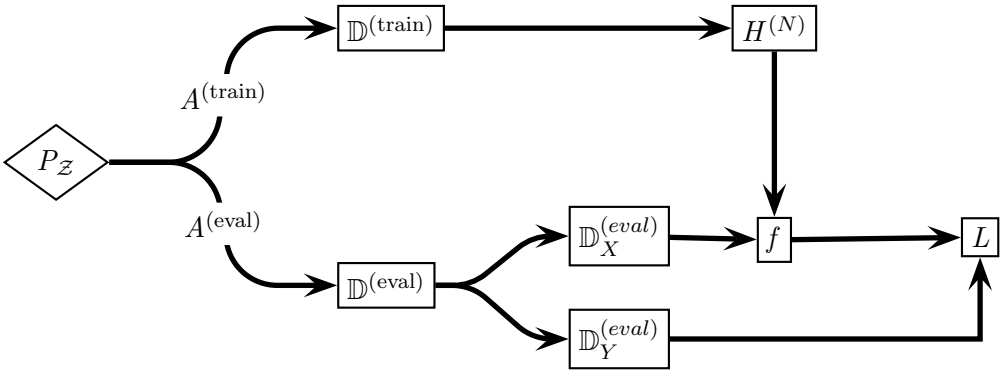


Figure 3.2: Diagram of a Causative attack against a learning system (see Figure 2.1).

1. **Defender** Choose procedure $H^{(N)}$ for selecting hypothesis
2. **Attacker** Choose procedures $A^{(\text{train})}$ and $A^{(\text{eval})}$ for selecting distributions
3. Evaluation:
   - Compute $P_{\mathcal{Z}}^{(\text{train})} \leftarrow A^{(\text{train})}\left(P_{\mathcal{Z}}, H\right)$
   - Sample dataset $\mathbb{D}^{(\text{train})}$ from $P_{\mathcal{Z}}^{(\text{train})}$
   - Compute $f \leftarrow H^{(N)}\left(\mathbb{D}^{(\text{train})}\right)$
   - Compute $P_{\mathcal{Z}}^{(\text{eval})} \leftarrow A^{(\text{eval})}\left(\mathbb{D}^{(\text{train})}, f\right)$
   - Sample dataset $\mathbb{D}^{(\text{eval})}$ from $P_{\mathcal{Z}}^{(\text{eval})}$
   - Assess total cost: $\sum_{\langle x,y\rangle \in \mathbb{D}^{(\text{eval})}} L_x\left(f\left(x\right), y\right)$

This game is very similar to the *Exploratory* game, but the attacker can choose $A^{(\text{train})}$ to affect the training data $\mathbb{D}^{(\text{train})}$. The attacker may have various types of influence over the data, ranging from arbitrary control over some fraction of instances to a small biasing influence on some aspect of data production; details depend on the setting. Again, the loss function $L_x\left(\cdot,\cdot\right)$ allows for instance-dependent costs.

Control over data used for training opens up new strategies to the attacker. Cost is based on the interaction of f and $\mathbb{D}^{(\text{eval})}$. In the *Exploratory* game the attacker chooses $\mathbb{D}^{(\text{eval})}$ while the defender controls f; in the *Causative* game the attacker also has influence on f. With this influence, the attacker can proactively cause the learner to produce bad classifiers.

Contamination in PAC learning: Kearns and Li [1993] extend Valiant's *probably ap-*

proximately correct (PAC) learning framework (*cf.*, Valiant [1984, 1985]) to prove bounds for maliciously chosen errors in the training data. In PAC learning, an algorithm succeeds if it can, with probability at least $1-\delta$, learn a hypothesis that has at most probability ϵ of making an incorrect prediction on an example drawn from the same distribution. Kearns and Li examine the case where an attacker has arbitrary control over some fraction β of the training examples (this specifies the form that $A^{(\text{train})}$ takes in our *Causative* game). They prove that in general the attacker can prevent the learner from succeeding if $\beta \geq \epsilon/(1+\epsilon)$, and for some classes of learners they show this bound is tight.

This work provides an interesting and useful bound on the ability to succeed at PAC-learning. The analysis broadly concerns both *Integrity* and *Availability* attacks as well as both *Targeted* and *Indiscriminate* variants. However, not all learning systems fall into the PAC-learning model.

3.5.2 Causative Integrity Attacks

In these attacks, the adversary actively attempts to corrupt the learning mechanism so that miscreant activities can take place that would be otherwise disallowed. In a *Causative Integrity* attack, the attacker uses control over training to cause intrusions to slip past the classifier as false negatives.

Example 3.3 (The Intrusion Foretold)
An attacker wants the defender's IDS not to flag a novel virus. The defender trains periodically on network traffic, so the attacker sends non-intrusion traffic that is carefully chosen to look like the virus and mis-train the learner to fail to block it. This example would be *Targeted* if the attacker already has a particular virus executable to send and needs to cause the learner to miss that particular instance. It would be *Indiscriminate*, on the other hand, if the attacker has a certain payload but could use any of a large number of existing exploit mechanisms to transmit the payload, in which case the attack need only fool the learner on any one of the malicious executables.

Red herring attack: Newsome et al. [2006] present *Causative Integrity* and *Causative Availability* attacks against Polygraph [Newsome et al., 2005], a polymorphic virus detector that learns virus signatures using both a conjunction learner and a naive-Bayes-like learner. Their *red herring* attacks against conjunction learners exploit certain weaknesses not present in other learning algorithms. The attack introduces spurious features along with their payload; once the learner constructs a signature, the spurious features are discarded to avoid subsequent detection. The idea is that the attacker transforms $P_{\mathcal{Z}}$ into $P_{\mathcal{Z}}^{(\text{train})}$ and $P_{\mathcal{Z}}^{(\text{eval})}$ to introduce spurious features into all malicious instances that the defender uses for training. The malicious instances produced by $P_{\mathcal{Z}}^{(\text{eval})}$, however, lack the spurious features and therefore bypass the filter, which erroneously generalized that the spurious features were necessary elements of the malicious behavior. Venkataraman et al. [2008] also present lower bounds for learning worm signatures based on red herring attacks.

Antidote: I also collaborated with colleagues at Berkeley and Intel Labs to explore the vulnerability of network-wide traffic anomaly detectors based on principal component analysis (PCA) as introduced by Lakhina et al. [2004b]. Our work examines how an attacker

can exploit the sensitivity of PCA to form *Causative Integrity* attacks [Rubinstein et al., 2009a]. In anticipation of a DoS attack, the attacker systematically injects traffic to increase variance along the links of their target flow and mislead the anomaly detection system. I also studied how the projection pursuit-based robust PCA algorithm of Croux et al. [2007] significantly reduces the impact of poisoning. I detail this work in Chapter 5.

3.5.3 Causative Availability Attacks

This less expected attack attempts to corrupt the learning system to cause normal traffic to significantly be misclassified to disrupt normal system operation. In a *Causative Availability* attack, the attacker uses control over training instances to interfere with operation of the system, such as by blocking legitimate traffic.

Example 3.4 (The Rogue IDS)
An attacker uses an intrusion detection system (IDS) to disrupt operations on the defender's network. The attacker wants traffic to be blocked so the destination doesn't receive it. The attacker generates attack traffic similar to benign traffic when the defender is collecting training data to train the IDS. When the learner re-trains on the attack data, the IDS will start to filter away benign instances as if they were intrusions. This attack could be *Targeted* at a particular protocol or destination. On the other hand, it might be *Indiscriminate* and attempt to block a significant portion of all legitimate traffic.

Allergy attack: Chung and Mok [2006, 2007] present *allergy* attacks against the Autograph worm signature generation system [Kim and Karp, 2004]. Autograph operates in two phases. First, it identifies infected nodes based on behavioral patterns, in particular scanning behavior. Second, it observes traffic from the identified nodes and infers blocking rules based on observed patterns. Chung and Mok describe an attack that targets traffic to a particular resource. In the first phase, an attack node convinces Autograph that it is infected by scanning the network. In the second phase, the attack node sends crafted packets mimicking targeted traffic, causing Autograph to learn rules that block legitimate access and create a denial of service event.

In the context of the *Causative* game, the attacker's choice of $P_{\mathcal{Z}}^{(\text{train})}$ provides the traffic for both phases of Autograph's learning. When Autograph produces a hypothesis f that depends on the carefully crafted traffic from the attacker, it will block access to legitimate traffic from $P_{\mathcal{Z}}^{(\text{eval})}$ that shares patterns with the malicious traffic.

Correlated outlier attack: Newsome et al. [2006] also suggest a *correlated outlier* attack against the Polygraph virus detector [Newsome et al., 2005]. This attack targets the naive-Bayes-like component of the detector by adding spurious features to positive training instances, causing the filter to block benign traffic with those features. As with the red herring attacks, these correlated outlier attacks fit neatly into the *Causative* game; this time $P_{\mathcal{Z}}^{(\text{train})}$ includes spurious features in malicious instances, causing $H^{(N)}$ to produce a f that classifies many benign instances as malicious.

Attacking SpamBayes: In the spam filtering domain I also explored *Causative Availability* attacks against the SpamBayes statistical spam classifier [Nelson et al., 2008, 2009].

In these attacks, I demonstrated that by sending emails containing entire dictionaries of tokens, the attacker can cause a significant fraction of normal email to be misclassified as spam with relatively little contamination (an *Indiscriminate* attack). Similarly, if an attacker can anticipate a particular target message, the attacker can also poison the learner to misclassify the target as spam (a *Targeted* attack). I also explored a principled defense to counter these *dictionary attacks*: the *reject on negative impact (RONI) defense*. I discuss this work in detail in Chapter 4.

3.5.4 Defending against Causative Attacks

Most defenses presented in the literature of secure learning combat *Exploratory Integrity* attacks (as discussed above) while relatively few defenses have been presented to cope with *Causative* attacks. In *Causative* attacks, the attacker has a degree of control over not only the evaluation distribution but also the training distribution. Therefore the learning procedures we consider must be resilient against contaminated training data, as well as to the evaluation considerations discussed in Section 3.4.4.

Two general strategies for defense are to remove malicious data from the training set and to harden the learning algorithm against malicious training data. I first present one method for the former and then describe two approaches to the latter that appear in the literature. The foundations of these approaches primarily lie in adapting game-theoretic techniques to analyze and design resilient learning algorithms.

3.5.4.1 The RONI defense

Insidious *Causative* attacks make learning inherently more difficult. In many circumstances, data sanitization may be the only realistic mechanism to achieve acceptable performance. For example, Nelson et al. [2009] introduce such a sanitization technique called the *Reject On Negative Impact (RONI) defense*, a technique that measures the empirical effect of adding each training instance and discards instances that have a substantial negative impact on classification accuracy. To determine whether a candidate training instance is malicious or not, the defender trains a classifier on a base training set, then adds the candidate instance to the training set and trains a second classifier. The defender applies both classifiers to a *quiz set* of instances with known labels and measures the difference in accuracy between the two classifiers. If adding the candidate instance to the training set causes the resulting classifier to produce substantially more classification errors, the defender permanently removes the instance as detrimental in its effect. I refine and explore the RONI defense experimentally in Section 4.5.5.

3.5.4.2 Learning with Contaminated Data

Several approaches to learning under adversarial contamination have been studied in the literature. The effect of adversarial contamination on the learner's performance were incorporated into some existing learning frameworks. Kearns and Li [1993] extended Valiant's *probably approximately correct* (PAC) model to allow for adversarial noise within the training data and bounded the amount of contamination a learner could tolerate. Separately, the field of *robust statistics* [see Huber, 1981, Hampel et al., 1986, Maronna et al., 2006]

formalized adversarial contamination with a worst-case contamination model from which analysts derived criteria for designing and comparing the robustness of statistical procedures to adversarial noise. Recent research incorporated these robustness criteria with more traditional learning domains [Christmann and Steinwart, 2004, Wagner, 2004], but generally these techniques have not been widely incorporated within machine learning. I discuss this further in the next section.

Another model of adversarial learning is based on the online expert learning setting [Cesa-Bianchi and Lugosi, 2006]. Rather than designing learners to be robust against adversarial contamination, techniques here focus on regret minimization to construct aggregate learners that adapt to adversarial conditions. The objective of regret minimization techniques is to dynamically aggregate the decisions of several experts based on their past performance so that the composite learner does well with respect to the best expert in hindsight; a set of techniques that I further discuss in Section 3.6.

3.5.4.3 Robustness

The field of robust statistics explores procedures that limit the impact of a small fraction of deviant (adversarial) training data. In the setting of robust statistics, it is assumed that the bulk of the data is generated from a known well-behaved model, but a fraction of the data comes from an unknown model—to bound the effect of this unknown source it is assumed to be adversarial. There are a number of measures of a procedure's robustness: the *breakdown point* is the level of contamination required for the attacker to arbitrarily manipulate the procedure and the *influence function* measures the impact of contamination on the procedure. Robustness measures can be used to assess the susceptibility of an existing system and to suggest alternatives that reduce or eliminate the vulnerability. Ideally one would like to use a procedure with a high breakdown point and a bounded influence function. These measures can be used to compare candidate procedures and to design procedures $H^{(N)}$ that are optimally robust against adversarial contamination of the training data. Here I summarize these concepts, but for a full treatment of these topics, refer to the books by Huber [1981], Hampel et al. [1986], and Maronna et al. [2006].

To motivate applications of robust statistics for adversarial learning, recall the traditional learning framework presented in Chapter 2.2. Particularly, in Chapter 2.2.4, I discussed selecting a hypothesis that minimizes the empirical risk. Unfortunately, in an adversarial setting, assumptions of the learning model may be violated. Ideally, one would hope that minor deviations from the modeling assumptions would not have a large impact on the optimal procedures that were derived under those assumptions. Unfortunately, this is not the case—small (adversarial) deviations from the assumptions can have a profound impact on some learning procedures. As stated by Tukey [1960]:

> A tacit hope in ignoring deviations from ideal models was that they would not matter; that statistical procedures which were optimal under the strict model would still be approximately optimal under the approximate model. Unfortunately, it turned out that this hope was often drastically wrong; even mild deviations often have much larger effects than were anticipated by most statisticians.

These flaws can also be exploited by an adversary to mistrain a learning algorithm even

when limited to a small amount of contamination. To avoid such vulnerabilities, one must augment the notion of optimality to include some form of *robustness* to the assumptions of the model; as defined by Huber [1981], "robustness signifies insensitivity to small deviations from the assumptions." There is, however, a fundamental trade-off between the efficiency of a procedure and its robustness—this issue is addressed in the field of robust statistics.

The model used to assess the distributional robustness of a statistical estimator H is known as the *gross-error model*, which is a mixture of the known distribution $F_{\mathcal{Z}}$ and some unknown distribution $G_{\mathcal{Z}}$ parameterized by some the fraction of contamination ϵ,

$$\mathcal{P}_\epsilon\left(F_{\mathcal{Z}}\right) \triangleq \{(1-\epsilon)F_{\mathcal{Z}} + \epsilon G_{\mathcal{Z}} \mid \ H_{\mathcal{Z}} \in \mathcal{P}_{\mathcal{Z}}\}$$

where $\mathcal{P}_{\mathcal{Z}}$ is the collection of all probability distributions on $\mathcal{Z}$. This concept of a contamination neighborhood provides for the *minimax approach* to robustness by considering a worst-case distribution within the gross-error model. Historically, the minimax approach yielded a robust class of estimators known as *Huber estimators*. Further it introduced the concept of a breakdown point ϵ^*—intuitively, the smallest level of contamination where the minimax asymptotic bias of an estimator becomes infinite.

An alternative approach is to consider the (scaled) change in the estimator H due to an infinitesimal fraction of contamination. Again, consider the gross-error models and define a derivative in the direction of an infinitesimal contamination localized at a single point z. By analyzing the scaled change in the estimator due to the contamination, one can assess the *influence* that adding contamination at point z has on the estimator. This gives rise to a functional known as the influence function and is defined as

$$\mathrm{IF}\left(z; H, F_{\mathcal{Z}}\right) \triangleq \lim_{\epsilon \to 0} \frac{H\left((1-\epsilon)F_{\mathcal{Z}} + \epsilon\Delta_z\right) - H\left(F_{\mathcal{Z}}\right)}{\epsilon}$$

where Δ_z is the distribution which has all its probability mass at the point z. This functional was derived for a wide variety of estimators and gives rise to several (infinitesimal) notions of robustness. The most prominent of these measures is the *gross-error sensitivity* defined as

$$\gamma^*\left(H, F_{\mathcal{Z}}\right) \triangleq \sup_z \left|\mathrm{IF}\left(z; H, F_{\mathcal{Z}}\right)\right| \ .$$

Intuitively, a finite gross error sensitivity gives a notion of robustness to infinitesimal point contamination.

Recent research has highlighted the importance of robust procedures in security and learning tasks. Wagner [2004] observes that common sensor net aggregation procedures, such as computing a mean, are not robust to adversarial point contamination, and he identifies robust alternatives as a defense against malignant or failed sensors. Christmann and Steinwart [2004] study robustness for a general family of learning methods. Their results suggest that certain commonly used loss functions, along with proper regularization, lead to robust procedures with a bounded influence function. These results suggest such procedures have desirable properties for secure learning, which I return to in Chapter 7.1.

3.6 Repeated Learning Games

In Section 3.4.1 and 3.5.1, the learning games are *one-shot games*, in which the defender and attacker minimize their cost when each move happens only once. Here, I generalize

these games to an *iterated game*, in which the players make a series of moves to minimize their total accumulated cost. I assume players have access to all information from previous iterations of the game.

In this setting, the defender can dynamically adapt to the adversary in an *online* fashion engendering a repeated game between the adversary and defender. The attacker has unspecified (potentially arbitrary) control of the training data, but instead of attempting to learn on this arbitrarily corrupted data, the online learner forms a composite prediction based on the advice of a set of M *experts* (*e.g.*, a set of classifiers each designed to provide different security properties). The game now takes place over K repetitions of the iterated *Causative* game. At each iteration, the experts provide advice (predictions) to the defender who weighs the advice of the experts to produce a composite prediction; *e.g.*, the aggregate prediction could be a weighted majority of the experts' predictions [Littlestone and Warmuth, 1994]. Further, at the end of the iteration, the defender learns the true labels for the predictions it made and it reweighs each expert based on the expert's prediction performance. No assumption is made about how the expert's form their advice or about their performance; in fact, their advice may be adversarial and may incur arbitrary loss. Rather than evaluating the cost of the composite predictions directly, one instead compares the cost incurred by the composite classifier relative to the cost of the best expert in hindsight; *i.e.*, we compute the *regret* that the composite classifier has for not heeding the advice of the best expert in hindsight. By using algorithms with small regret, the composite predictor performs comparably to the best expert without knowing which one will be best, *a priori*. Thus, by designing strategies that minimize regret, online learning provides an elegant mechanism to combine several predictors, each designed to address the security problem in a different way, into a single predictor that adapts relative to the performance of its constituents. As a result, the attacker must design attacks that are uniformly successful on the set of predictors rather than just on a single predictor because the composite learner can perform almost as well as the best without knowing ahead of time which expert will be best. A full description of this setting and several regret minimization learning algorithms appear in Cesa-Bianchi and Lugosi [2006].

In this setting, the learner forms a prediction from the M expert predictions and adapts its predictor $h^{(k)}$ based on their performance during K repetitions. At each step k of the game, the defender receives a prediction $\hat{y}^{(k,m)}$ from the m^{th} expert[2] and make a composite prediction $\hat{y}^{(k)}$ via $h^{(k)}$. After the defender's prediction is made, the true label $y^{(k)}$ is revealed and the defender evaluates the *instantaneous regret* for each expert; *i.e.*, the difference in the loss for the composite prediction and the loss for the m^{th} expert's prediction. More formally, the k^{th} round of the expert-based prediction game is[3]:

[2]An expert's advice may be based on the data but the defender makes no assumption about how experts form their advice.

[3]Here, I again assume that costs are symmetric for the defender and adversary and are represented by loss function. Further, as in Chapter 2.2.4 I simplify the game to use ignore the surrogate loss function used in place of 0/1 losses. Finally, this game is also easily generalized to the case where several instances/labels are generated in each round of the game.

1. **Defender** Update function $h^{(k)} : \mathcal{Y}^M \to \mathcal{Y}$
2. **Attacker** Choose distribution $P_{\mathcal{Z}}^{(k)}$
3. Evaluation:
   - Sample an instance $\left(x^{(k)}, y^{(k)}\right) \sim P_{\mathcal{Z}}^{(k)}$
   - Compute expert advice $\left\{\hat{y}^{(k,m)}\right\}_{m=1}^{M}$; *e.g.*, $\hat{y}^{(k,m)} = f^{(m)}\left(x^{(k)}\right)$
   - Predict $\hat{y}^{(k)} = h^{(k)}\left(\hat{y}^{(k,1)}, \hat{y}^{(k,2)}, \ldots, \hat{y}^{(k,M)}\right)$
   - Compute instantaneous regret: $r^{(k,m)} = L\left(\hat{y}^{(k)}, y^{(k)}\right) - L\left(\hat{y}^{(k,m)}, y^{(k)}\right)$ for each expert $m = 1 \ldots M$

This game has a slightly different structure from the games I presented in Section 3.4.1 and 3.5.1—here the defender chooses one strategy at the beginning of the game and then in each iteration updates the function $h^{(k)}$ according to that strategy. Based only on the past performance of each expert (*i.e.* the regrets observed over the previous $k-1$ iterations of the game), the defender chooses an online strategy for updating $h^{(k)}$ at the k^{th} step of the game to minimize regret [*cf.*, Cesa-Bianchi and Lugosi, 2006]. The attacker, however, may select a new strategy at each iteration and can control the subsequent predictions made by each expert based on the defender's choice for $h^{(k)}$.

Finally, at the end of the game, the defender is assessed in terms of the regret for the predictions it made. At each iteration the defender would like to choose the best advice given at that iteration, but that is not possible since, in the worst-case, the adversary is assumed to choose the advice given by each expert. Instead, the overall performance of the defender is compared to the overall performance of each expert through the defender's *cumulative regret*; *i.e.*, the cumulative difference between the loss of the composite learner and the loss of the m^{th} expert. The cumulative regret $R^{(m)}$ for the composite predictor with respect to the m^{th} expert and the *worst-case regret* over all experts are thus defined as

$$R^{(m)} \triangleq \sum_{k=1}^{K} r^{(k,m)} \qquad\qquad R^{*} \triangleq \max_{m} R^{(m)} \tag{3.1}$$

If R^* is small (relative to K), then the defender's aggregation algorithm has performed almost as well the best expert without knowing which expert would be best. Further, as follows from the Equation (3.1) and the definition of instantaneous regret, the average regret is simply the difference of the risk of $h^{(k)}$ and the risk of $f^{(m)}$. Thus, if the average worst-case regret is small (*i.e.*, approaches 0 as K goes to infinity) and the best expert has small risk, the predictor $h^{(k)}$ also has a small risk. This motivates the study of *regret minimization procedures*. A substantial body of research has explored strategies for choosing $h^{(k)}$ to minimize regret in several settings.

Online expert-based prediction splits risk minimization into two subproblems: (*i*) minimizing the risk of each expert, and (*ii*) minimizing the average regret; that is, as if we had known the best predictor $f^{(*)}$ before the game started and had simply used its prediction at every step of the game. The other defenses we have discussed approach the first problem. Regret minimization techniques address the second problem: the defender chooses a strategy for updating $h^{(k)}$ to minimize regret based only on the expert's past performance.

For certain variants of the game, there exist composite predictors whose regret is $o(K)$—that is, the average regret approaches 0 as K increases. Thus, the composite learner can perform almost as well as the best expert without knowing ahead of time which expert is best. Hence, if there is any single predictor that predicted well, the combined predictor will predict nearly as well. This effectively allows the defender to use several strategies simultaneously and forces the attacker to design attacks that do well against them all.

Importantly, regret minimization techniques allow the defender to adapt to an adversary and force the adversary to design attack strategies that succeed against an entire set of experts (each of which can have its own security design considerations and may use different feature sets, different hypothesis spaces, or different training procedures). Thus, one can incorporate several classifiers with desirable security properties into a composite approach. Moreover, if a successful attack is discovered, one can design a new expert against the identified vulnerability and add it to our set of experts to patch the exploit. This makes online prediction well-suited to the ever-changing attack landscape.

3.7 Dissertation Organization

I partition the remainder of my dissertation work based on the framework presented in this chapter. I divide my research into two parts. The first explores *Causative* attacks while the second examines *Exploratory* attacks. Incidentally, the first part is primarily concerned with analyzing the security of real systems while the second part deals with theoretical questions of classifier evasion.

The next part of my dissertation investigates *Causative* attacks against two practical learning systems. In the first, I analyze a spam filter called SpamBayes and show that it is particularly vulnerable to *Availability* attacks through adversarial contamination of the training data. The adversary's contamination model uses data insertion to inject a number of attack *spam* messages into the filter's training set. I propose a data sanitization defense that is able to successfully detect and remove attack messages based on the estimated damage the message causes. The second learning system I analyze is a network anomaly detection system based on a subspace estimation technique (principal component analysis). For this system, the adversary instead undertakes *Integrity* attacks and the adversary uses a data alteration model to contaminate the training set. Also, to combat attacks against these detectors, I propose an alternate learning approach based on a technique from robust statistics.

In the final part of my dissertation, I examine an important theoretical model for *Exploratory* attacks against a classifier. To find a classifier's blind spots the adversary systematically issues membership queries and uses the classifier's responses to glean important structural information about its boundary. I generalize this framework, first presented by Lowd and Meek [2005b], to a more diverse family of classifiers called the convex-inducing classifiers and to a broader set of ℓ_p distances. Further, in investigating the near-optimal evasion problem, I suggest a number of novel research directions to pursue within the *Exploratory* attack setting.

Part I

Protecting against False Positives and False Negatives in Causative Attacks:

Two Case Studies of Availability and Integrity Attacks

Chapter 4

Availability Attack Case Study: SpamBayes

Adversaries can launch *Causative Availability* attacks that result in classifiers that have unacceptably high false positive rates; *i.e.*, that misclassify benign input as potential attacks causing undue interruption in legitimate activity. This chapter provides a case study of one such attack on the SpamBayes spam detection system. I show that cleverly-crafted attack messages—pernicious spam email that an uninformed human user would likely identify and label as spam—can exploit SpamBayes' learning algorithm causing the resulting classifier to have an unreasonably high false positive rate[1]. I also show effective defenses against these attacks and discuss the trade-offs required to prevent them.

I examine several attacks against the SpamBayes spam filter each of which embodies a particular insight into the vulnerability of the underlying learning technique. In doing so, I more broadly demonstrate attacks that could impact any system that uses a similar learning algorithm. Most notably, the attacks I present in this chapter target the learning algorithm used by the spam filter SpamBayes (spambayes.sourceforge.net), but several other filters also use the same underlying learning algorithm; this includes BogoFilter (bogofilter.sourceforge.net), the spam filter in Mozilla's Thunderbird email client (mozilla.org), and the machine learning component of SpamAssassin (spamassassin.apache.org). The primary difference between the learning elements of these three filters is in their tokenization methods; *i.e.*, the learning algorithm is fundamentally identical but each filter uses a different set of features. I demonstrate the vulnerability of the underlying algorithm for SpamBayes because it uses a pure machine learning method, it is familiar to the academic community Meyer and Whateley [2004], and it is popular with over 700,000 downloads. Although here I only analyze SpamBayes, the fact that these other systems use the same learning algorithm suggests that other filters are also vulnerable to similar attacks. However, the overall effectiveness of the attacks would depend on how each of the other filters incorporated the learned classifier into the final filtering decision. For instance, filters such as SpamAssassin, only use learning as one of several components of a broader filtering engine (the others are hand-crafted non-adapting rules), so attacks against it would degrade the performance of the filter but perhaps the overall impact would be lessened or muted en-

[1]Chapter 5 also demonstrates *Causative* attacks that instead result in classifiers with an unreasonably high false negative rate—these are *Integrity* attacks.

tirely. In principle, though, it should be possible to replicate these results in these other filters. Finally, beyond spam filtering, I highlight the vulnerabilities in SpamBayes' learner because these same attacks could also be employed against similar learning algorithms in other domains. While the feasibility of these attacks, the attacker's motivation, or the contamination mechanism present in this chapter may not be appropriate in other domains, it is nonetheless interesting to understand the vulnerability so that it can be similarly assessed for other applications.

I organize my approach to studying the vulnerability of SpamBayes' learning algorithm based on the framework discussed in Chapter 3. Primarily, I investigated *Causative Availability* attacks on the filter as this type of attack was an interesting new facet to attacks against a learner that could actually be deployed in real-world settings. The adversary I studied has an additive contamination capability (*i.e.*, the adversary has exclusive control on some subset of the user's training data) but limited to only altering the positive (spam) class; I deemed this contamination model to be the most appropriate for a crafty spammer. Novel contributions of my research include a set of successful principled attacks against SpamBayes, an empirical study validating the effectiveness of the attacks in a realistic setting, and a principled defense that empirically succeeds against several of the attacks. I finally discuss the implications of the attack and defense strategies and the role that attacker information plays in the effectiveness of their attacks.

Below, I discuss the background of the training model (see Section 4.1); I present three new attacks on SpamBayes (see Section 4.3); I give experimental results (see Section 4.5); and I present a defense against these attacks together with further experimental results (see Section 4.4). This work appeared in the *First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)* [Nelson et al., 2008] and was subsequently published as a book chapter in *Machine Learning in Cyber Trust: Security, Privacy, Reliability* [Nelson et al., 2009].

4.1 The SpamBayes Spam Filter

SpamBayes is a content-based statistical spam filter that classifies email using token counts in a model proposed by Robinson [2003] as inspired by Graham [2002]. Meyer and Whateley [2004] describe the system in detail. SpamBayes computes a spam score for each token in the training corpus based on its occurrence in spam and non-spam emails; this score is motivated as a smoothed estimate of the posterior probability that an email containing that token is spam. The filter computes a message's overall spam score based on the assumption that the token scores are independent and then it applies Fisher's method [see Fisher, 1948] for combining significance tests to determine whether the email's tokens are sufficiently indicative of one class or the other. The message score is compared against two thresholds to select the label *spam*, *ham* (*i.e.*, non-spam), or *unsure*. In the remainder of this section, I detail the statistical method SpamBayes uses to estimate and aggregate token scores.

4.1.1 SpamBayes' Training Algorithm

SpamBayes is a content-based spam filter that classifies messages based on the tokens (including header tokens) observed in an email. The spam classification model used by SpamBayes was designed by Robinson [2003] and Meyer and Whateley [2004], based on ideas by Graham [2002] together with Fisher's method for combining independent significance tests [Fisher, 1948]. Intuitively, SpamBayes learns how strongly each token indicates *ham* or *spam* by counting the number of each type of email that token appears in. When classifying a new email, SpamBayes considers all the message's tokens as evidence of whether the message is spam or ham and uses a statistical test to decide whether they indicate one label or the other with sufficient confidence; if not, SpamBayes returns *unsure*.

SpamBayes tokenizes each email X based on words, URL components, header elements, and other character sequences that appear in X. Each is treated as a unique token of the email independent of their order within the message and for convenience, I place an ordering on the tokens to name a unique token as the i^{th} token (among the entire alphabet of tokens). Further, SpamBayes only records whether or not a token occurs in the message, not how many times it occurs. Email X is represented as a binary (potentially infinite length) vector $\mathbf{x}$ where

$$x_i = \begin{cases} 1\,, & \text{if the } i^{\text{th}} \text{ token occurs in } X \\ 0\,, & \text{otherwise} \end{cases} .$$

This message vector representation records which tokens occur in the message independent of their order or multiplicity.

The training data used by SpamBayes is a dataset of message vector (representing each training message) and label pairs: $\mathbb{D}^{(\text{train})} = \left\{\left\langle \mathbf{x}^{(1)}, y^{(1)} \right\rangle, \left\langle \mathbf{x}^{(2)}, y^{(2)} \right\rangle, \ldots, \left\langle \mathbf{x}^{(N)}, y^{(N)} \right\rangle\right\}$ where $\mathbf{x}^{(i)} \in \{0,1\}^D$ and $y^{(i)} \in \{ham, spam\}$. As in Section 2.2.1, this training data can be represented as a training matrix $\mathbf{X} = \left[\mathbf{x}^{(1)}\, \mathbf{x}^{(2)}\, \ldots\, \mathbf{x}^{(N)}\right]^\top \in \{0,1\}^{N \times D}$ along with its label vector $\mathbf{y} = \left[y^{(1)}\, y^{(2)}\, \ldots\, y^{(N)}\right] \in \{ham, spam\}^N$. Using the training matrix, the token-counting statistics used by SpamBayes can be expressed as

$$\begin{aligned} \mathbf{n}^{(s)} &\triangleq \mathbf{X}^\top \mathbf{y} \\ \mathbf{n}^{(h)} &\triangleq \mathbf{X}^\top (\mathbf{1} - \mathbf{y}) \\ \mathbf{n} &\triangleq \mathbf{n}^{(s)} + \mathbf{n}^{(h)} \end{aligned}$$

which are vectors containing the cumulative token counts for each token in all, spam, and ham messages respectively. I also define $N^{(s)} \triangleq \mathbf{y}^\top \mathbf{y}$ as the total number of training spam messages and $N^{(h)} \triangleq (\mathbf{1} - \mathbf{y})^\top (\mathbf{1} - \mathbf{y})$ as the total number of training ham messages (and, of course, $N = N^{(s)} + N^{(h)}$).

From these count statistics, SpamBayes computes a spam score for the i^{th} token by estimating the posterior $\Pr(X \text{ is spam} | x_i = 1)$. First, the likelihoods $\Pr(x_i = 1 | X \text{ is spam})$ and $\Pr(x_i = 1 | X \text{ is ham})$ for observing the i^{th} token in a spam/ham message are estimated using the maximum likelihood estimators yielding the likelihood vectors $L_i^{(s)} = \frac{1}{N^{(s)}} \cdot \mathbf{n}^{(s)}$ and $L_i^{(h)} = \frac{1}{N^{(h)}} \cdot \mathbf{n}^{(h)}$.

Second, using the likelihood estimates $\mathbf{L}^{(s)}$ and $\mathbf{L}^{(h)}$ and an estimate $\pi^{(s)}$ on the *prior distribution* $\Pr(X \text{ is spam})$, Bayes' Rule is used to estimate the posteriors as $\mathbf{P}^{(s)} \propto \frac{\pi^{(s)}}{N^{(s)}} \cdot \mathbf{n}^{(s)}$

and $\mathbf{P}^{(h)} \propto \frac{1-\pi^{(s)}}{N^{(h)}} \cdot \mathbf{n}^{(h)}$ along with the constraints $\mathbf{P}^{(s)} + \mathbf{P}^{(h)} = \mathbf{1}$. However, instead of using the usual naive Bayes maximum likelihood prior estimator $\pi^{(s)} = \frac{N^{(s)}}{N^{(s)}+N^{(h)}}$, SpamBayes uses the agnostic prior $\pi^{(s)} = \frac{1}{2}$; a choice that gives their learner unusual properties which I discuss further in Appendix B.2.1. Nonetheless, based on this choice of prior, SpamBayes computes a *spam score vector* $\mathbf{P}^{(s)}$ specified for the i^{th} token as

$$P_i^{(s)} \quad = \quad \frac{N^{(h)} n_i^{(s)}}{N^{(h)} n_i^{(s)} + N^{(s)} n_i^{(h)}} \ ; \tag{4.1}$$

i.e., an estimator of the posterior $\Pr(X \text{ is spam} | x_i = 1)$. An analogous *token ham score* is given by $\mathbf{P}^{(h)} = \mathbf{1} - \mathbf{P}^{(s)}$.

Robinson's method Robinson [2003] smooths $P_i^{(s)}$ through a convex combination with a prior belief x (default value of $x = 0.5$), weighting the quantities by n_i (the number of training emails with the i^{th} token) and s (chosen for strength of prior with a default of $s = 1$), respectively:

$$q_i = \frac{s}{s + n_i} \, x + \frac{n_i}{s + n_i} \, P_i^{(s)} \ . \tag{4.2}$$

Here, smoothing mitigates over estimation for rare tokens. For instance, if the token "floccinaucinihilipilification" appears once in a spam and never in a ham in the training set, the posterior estimate would be $P_i^{(s)} = 1$, which would make any future occurrence of this word dominate the overall spam score. However, occurrence of the word only in spam could have just been an artifact of the overall rarity of the word. In this case, smoothing is done by adding a prior that the posterior for every token is $x = \frac{1}{2}$ (*i.e.*, an agnostic score). For rare tokens, the posterior estimate is dominated by this prior. However, as more tokens are observed, the smoothed score approaches the empirical estimate of the posterior in Equation (4.1) according to the strength given to the prior by s. An analogous smoothed ham score is given by $1 - \mathbf{q}$.

4.1.2 SpamBayes' Prediction

After training, the filter computes the overall spam score $I(\hat{\mathbf{x}})$ of a new message $\hat{X}$ using Fisher's method [Fisher, 1948] for combining the scores of the tokens observed in $\hat{X}$. SpamBayes uses at most 150 tokens from $\hat{X}$ with scores furthest from 0.5 and outside the interval $(0.4, 0.6)$ (see Appendix B.2.2 for more details). Let $\mathbb{T}_{\hat{\mathbf{x}}}$ be the set of tokens that SpamBayes incorporates into its spam score and let $\boldsymbol{\delta}(\hat{\mathbf{x}})$ be the indicator function for this set. The token spam scores are combined into a *message spam score* for $\hat{X}$ by

$$S(\hat{\mathbf{x}}) = 1 - \chi^2_{2\tau_{\hat{\mathbf{x}}}} \left(-2(\log \mathbf{q})^\top \boldsymbol{\delta}(\hat{\mathbf{x}}) \right), \tag{4.3}$$

where $\tau_{\hat{\mathbf{x}}} \triangleq |\mathbb{T}_{\hat{\mathbf{x}}}|$ is the number of token from $\hat{X}$ used by SpamBayes and $\chi^2_{2\tau_{\hat{\mathbf{x}}}}(\cdot)$ denotes the cumulative distribution function of the chi-square distribution with $2\tau_{\hat{\mathbf{x}}}$ degrees of freedom. A ham score $H(\hat{\mathbf{x}})$ is similarly defined by replacing $\mathbf{q}$ with $1 - \mathbf{q}$ in Equation (4.3). Finally,

SpamBayes constructs an overall spam score for $\hat{X}$ by averaging $S(\hat{\mathbf{x}})$ and $1 - H(\hat{\mathbf{x}})$ (both being indicators of whether $\hat{X}$ is spam) giving the final score

$$I(\hat{\mathbf{x}}) = \frac{S(\hat{\mathbf{x}}) + 1 - H(\hat{\mathbf{x}})}{2} \tag{4.4}$$

for a message; a quantity between 0 (strong evidence of ham) and 1 (strong evidence of spam). SpamBayes predicts by thresholding $I(\hat{\mathbf{x}})$ against two user-tunable thresholds $\theta^{(h)}$ and $\theta^{(s)}$, with defaults $\theta^{(h)} = 0.15$ and $\theta^{(s)} = 0.9$. SpamBayes predicts *ham*, *unsure*, or *spam* if $I(\hat{\mathbf{x}})$ falls into the interval $\left[0, \theta^{(h)}\right]$, $\left(\theta^{(h)}, \theta^{(s)}\right]$, or $\left(\theta^{(s)}, 1\right]$, respectively, and filters the message accordingly.

The inclusion of an *unsure* label in addition to *spam* and *ham* prevents us from purely using *ham-as-spam* and *spam-as-ham* misclassification rates (false positives and false negatives, respectively) for evaluation. We must also consider *spam-as-unsure* and *ham-as-unsure* misclassifications. Because of the practical effects on the user's time and effort discussed in Section 4.2.3, *ham-as-unsure* misclassifications are nearly as bad for the user as *ham-as-spam*.

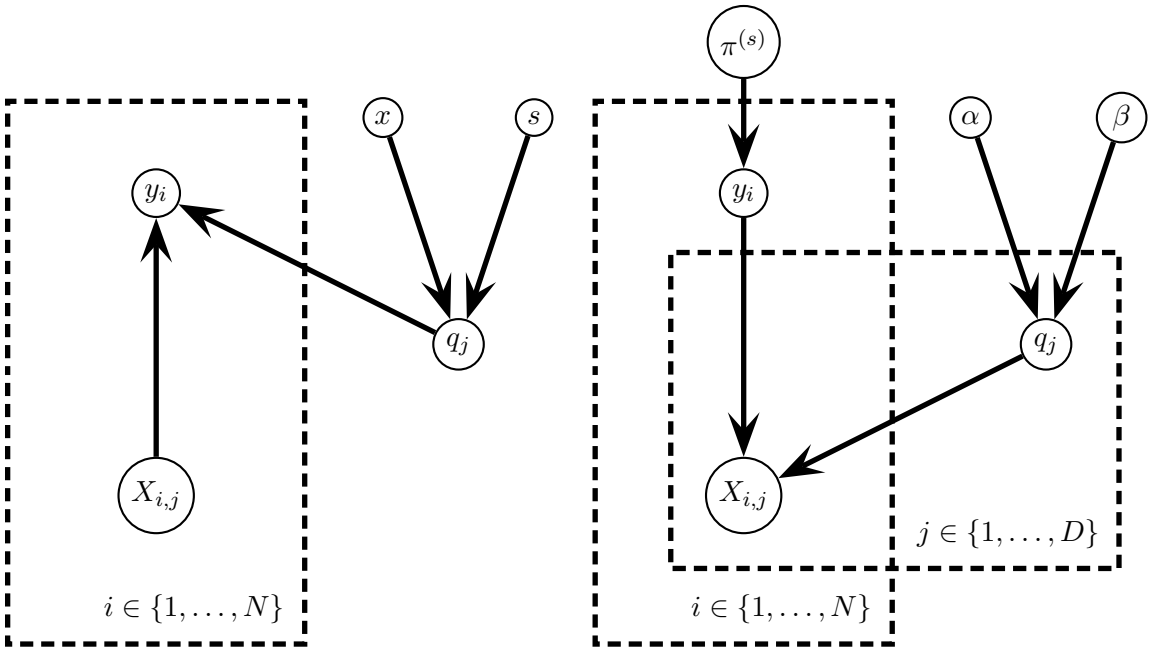
4.1.3 SpamBayes' Model

Although the components of SpamBayes algorithm (token spam scores, smoothing, and chi-squared test) were separately motivated, the resulting system can be described by a unified probability model for discriminating ham from spam messages. While Robinson motivates the SpamBayes classifier as a smoothed estimator of the posterior probability of spam, they never explicitly specify the probabilistic model. Here, I specify a discriminative model and show that the resulting estimation can be re-derived as using empirical risk minimization. Doing so provides a better understanding of the modeling assumptions of the SpamBayes classifier and its vulnerabilities.

In this model, there are three random variables of interest: the spam label y_i of the i^{th} message (Here I use the convention that this label is a 1 to indicate *spam* or a 0 to indicate *ham*), the indicator variable $X_{i,j}$ of the j^{th} token in the i^{th} message, and the token score q_j of the j^{th} token. In the discriminative setting, given $\mathbf{X}_{i,\bullet}$ as a representation of the tokens in the i^{th} message and the token scores $\mathbf{q}$, the message's label y_i is conditionally independent of all other random variables in the model. The conditional probability of the message label given the occurrence of a single token $X_{i,j}$ is specified by

$$\Pr(y_i | X_{i,j}, q_j) = \left((q_j)^{y_i} \cdot (1 - q_j)^{1 - y_i} \right)^{X_{i,j}} \left(\tfrac{1}{2}\right)^{1 - X_{i,j}} \quad , \tag{4.5}$$

i.e., in the SpamBayes model, each token that occurs in the message is an indicator of its label whereas tokens absent from the message have no impact on its label. Because SpamBayes' scores only incorporate tokens that occur in the message, traditional generative spam models (*e.g.*, Figure 4.1(b)) are awkward to construct, but the above discriminative conditional probability captures this modeling nuance. Further, there is no prior for the token indicators $X_{i,j}$ but there is a prior on the token scores. Treating these as binomial parameters, each has a *beta prior* with common parameters α and β giving them a conditional



(a) SpamBayes' Discriminative Model (b) Traditional Generative Model

Figure 4.1: Probabilistic graphical models for spam detection. **(a)** A probabilistic model that depicts the dependency structure between random variables in SpamBayes for a *single* token (SpamBayes models each token as a separate indicator of ham/spam and then combines them together assuming each is an independent test). In this model, the label y_i for the i^{th} email depends on the token score q_j for the j^{th} token if it occurs in the message; *i.e.*, $X_{i,j} = 1$. The parameters s and x parameterize a beta prior on q_j. **(b)** A more traditional generative model for spam. The parameters $\pi^{(s)}$, α, and β parameterize the prior distributions for y_i and q_j. Each label y_i for the i^{th} email is drawn independently from a Bernoulli distribution with $\pi^{(s)}$as the probability of *spam*. Each token score for the j^{th} token is drawn independently from a beta distribution with parameters α and β. Finally, given the label for a message and the token scores, $X_{i,j}$ is drawn independently from a Bernoulli. Based on the likelihood function for this model, the token scores q_j computed by SpamBayes can be viewed simply as the maximum likelihood estimators for the corresponding parameter in the model.

probability of

$$\Pr\left(q_j|\alpha,\beta\right) = \frac{1}{\mathrm{B}\left(\alpha,\beta\right)} \cdot \left(q_j\right)^{\alpha-1} \cdot \left(1-q_j\right)^{\beta-1} \quad , \tag{4.6}$$

where $\mathrm{B}\left(\alpha,\beta\right)$ is the *beta function*. As earlier mentioned, Robinson instead use an equivalent parameterization with a strength parameter s and prior parameter x, for which $\alpha = s \cdot x + 1$ and $\beta = s\left(1-x\right)+1$. Using this parameterization, x specifies the mode of the prior distribution. In SpamBayes, the prior parameters are fixed a priori rather than treated as random hyper-parameters (by default, these take the values $\pi^{(s)} = \frac{1}{2}$, $x = \frac{1}{2}$, and $s = 1$).

Together, the label's probability conditioned on the j^{th} token and the prior on the j^{th} token score are used to derive a spam score for the message (based only on the j^{th} token). However, unlike a maximum likelihood derivation, SpamBayes' parameter estimation for q_j is not based on a joint probability model over all tokens. Instead, the score for each token is computed separately by maximizing the labels' likelihood within a per-token token model as depicted in Figure 4.1(a); *i.e.*, the model depicts a sequence of labels based solely on the presence of the j^{th} token. Based on the independence assumption of Figure 4.1(a), the conditional distributions of Equation (4.5) combine together to make the following joint log probability based on the j^{th} token (for N messages):

$$\begin{aligned}
\log\Pr\left(\mathbf{y},\mathbf{X}_{\bullet,j}|\alpha,\beta\right) &= \log\Pr\left(q_j|\alpha,\beta\right) + \sum_{i=1}^{N}\log\Pr\left(y_i|X_{i,j},q_j\right) \\
&= -\log\left(\mathrm{B}\left(\alpha,\beta\right)\right) + \left(\alpha-1\right)\log\left(q_j\right) + \left(\beta-1\right)\log\left(1-q_j\right) \\
&\quad + \sum_{i=1}^{N}\left[y_i X_{i,j}\log\left(q_j\right) + \left(1-y_i\right)X_{i,j}\log\left(1-q_j\right)\right]
\end{aligned}$$

Maximizing this joint distribution (nearly) achieves the token scores specified by SpamBayes. To solve for maximum, differentiate the joint probability with respect to the j^{th} token score, q_j, and set the derivative equal to 0. This yields

$$\begin{aligned}
q_j &= \frac{\sum_{i=1}^{N} y_i X_{i,j} + \alpha - 1}{\sum_{i=1}^{N} X_{i,j} + \alpha - 1 + \beta - 1} \\
&= \frac{\alpha-1}{n_j + \alpha - 1 + \beta - 1} + \frac{n_j^{(s)}}{n_j + \alpha - 1 + \beta - 1} \quad ,
\end{aligned}$$

where the summations in the first equation are simplified to token counts based on the definitions of y_i and $X_{i,j}$. Using the equivalent beta parameterization with x and s and the usual posterior token score $P_i^{(s)} = \frac{n_i^{(s)}}{n_i^{(s)}+n_i^{(h)}}$ (which differs from the SpamBayes token score used in Equation (4.1) unless $N^{(s)} = N^{(h)}$), this equation for the maximum-likelihood estimator of q_j is equivalent to the SpamBayes' estimator in Equation (4.2).

The above per-token optimizations can also be viewed as a joint maximization procedure by considering the overall spam and ham scores $S\left(\cdot\right)$ and $H\left(\cdot\right)$ for the messages in the training set (see Equation 4.3). These overall scores are based on Fisher's method for combining independent p-values and assume that each token score is independent. In fact, $S\left(\cdot\right)$ and $H\left(\cdot\right)$ are tests for the aggregated scores $s_{\mathbf{q}}\left(\cdot\right)$ and $h_{\mathbf{q}}\left(\cdot\right)$ defined by Equations (B.1) and (B.2)—tests that monotonically increase with $s_{\mathbf{q}}\left(\cdot\right)$ and $h_{\mathbf{q}}\left(\cdot\right)$, respectively. Thus, from the overall spam score $I\left(\cdot\right)$ defined by Equation (4.4), maximizing $s_{\mathbf{q}}\left(\cdot\right)$ for all spam and

$h_{\mathbf{q}}(\cdot)$ for all ham is a surrogate for minimizing the prediction error of $I(\cdot)$; *i.e.*, minimizing some loss for $I(\cdot)$. Hence, combining the individual tokens' conditional distributions (Equation 4.5) together to form

$$Q(y_i, \mathbf{X}_{i,\bullet}, \mathbf{q}) = -\log \prod_{j=1}^{D} \left[(q_j)^{y_i} \cdot (1-q_j)^{1-y_i} \right]^{X_{i,j}} ,$$

can be viewed as the loss function for the score $I(\cdot)$ and the sum of the negative logarithm of the token score priors given by Equation 4.6 can be viewed as its regularizer[2]. Moreover, minimizing this *regularized empirical loss* again yields the SpamBayes' token scores from Equation (4.2). In this way, SpamBayes can be viewed as a regularized empirical risk minimization technique.

Unfortunately, the loss function Q above is not a negative log-likelihood because the product of the scores is unnormalized. When the proper normalizer is added to Q, the resulting parameter estimates for q_j no longer are equivalent to SpamBayes' estimators. In fact, SpamBayes' parameter estimation procedure and its subsequent prediction rule do not appear to be compatible with a traditional joint probability distribution over all labels, tokens, and scores (or at least I was unable to derive a joint probability model that would yield these estimates). It is unclear whether the SpamBayes loss function Q has a reasonable motivation or whether it is an appropriate loss function to use for spam detection.

Nonetheless, by analyzing the model of SpamBayes, I can now identify its potential vulnerabilities. First, by incorporating a prior on the token scores for smoothing, Robinson prevented a simple attack. Without any smoothing on the token scores, all tokens that only appear in ham would have token scores of 0. Since the overall score $I(\cdot)$ is computed with products of the individual token scores, including any of these ham-only tokens would cause spam to be misclassified as *ham* (and vice-versa for spam-only tokens), which the adversary could clearly exploit. Similarly, using the censor function $\mathbb{T}$ helps prevent attacks in which the adversary pads a spam with many *hammy* tokens to negate the effect of *spammy* tokens. However, despite these design considerations, SpamBayes is still vulnerable to attacks. The first vulnerability of SpamBayes comes from its assumption that the data and tokens are independent, for which each token score is estimated based solely on the presence of that token in ham and spam messages. The second vulnerability comes from its assumption that only tokens that occur in a message contribute to its label. While there is some intuition behind this assumption, in this model, it causes rare tokens to have little support so that their scores can be easily changed. Ultimately, these two vulnerabilities lead to a family of attacks that I call *dictionary attacks* that I present and evaluate in the rest of this chapter.

4.2 Threat Model for SpamBayes

In analyzing the vulnerabilities of SpamBayes, I was motivated by the taxonomy of attacks (*cf.*, Chapter 3.3). Known real-world attacks that spammers use against deployed spam filters tend to be *Exploratory Integrity* attacks: either the spammer obfuscates the especially

[2]This interpretation ignores the censoring function $\mathbb{T}$ in which SpamBayes only uses the scores of the most informative tokens when computing $I(\cdot)$ for a message. As discussed in Appendix B.1 this censoring action makes $I(\cdot)$ non-monotonic in the token scores q_j. Computing the token scores without considering $\mathbb{T}$ can be viewed as a tractable relaxation of the true objective.

spam-like content of a spam email or he includes content not indicative of spam. Both tactics aim to get the modified message into the victim's inbox. This category of attack has been studied in detail in the literature [*e.g.*, see Lowd and Meek, 2005a, Wittel and Wu, 2004, Lowd and Meek, 2005b, Dalvi et al., 2004]. However, I found the study of *Causative* attacks more compelling because they are unique to machine learning systems and potentially more harmful.

In particular, a *Causative Availability* attack can create a powerful denial of service. For example, if a spammer causes enough legitimate messages to be filtered by the user's spam filter, the user is likely to disable the filter and therefore see the spammer's advertisements. As another example, an unscrupulous business owner may wish to use spam filter denial of service to prevent a competitor from receiving email orders from potential customers. In this chapter, I present two novel *Causative Availability* attacks against SpamBayes: the dictionary attack is *Indiscriminate* and the focused attack is *Targeted*.

4.2.1 Attacker Goals

I consider an attacker with one of two goals: expose the victim to an advertisement or prevent the victim from seeing a legitimate message. The motivation for the first objective is obviously the potential revenue gain for the spammer if their marketing campaign is widely viewed. For the second objective, there are at least two motives for the attacker to cause legitimate emails to be filtered as spam. First, a large number of misclassifications will make the spam filter unreliable, causing users to abandon filtering and see more spam. Second, causing legitimate messages to be mislabeled can cause users to miss important messages. For example, an organization competing for a contract wants to prevent competing bids from reaching their intended recipient to gain a competitive advantage; an unscrupulous company can achieve this by causing their competitor's messages to be filtered as spam.

Based on these considerations, we can further divide the attacker's goals into four categories:

1. Cause the victim to *disable* the spam filter, thus letting all spam into the inbox
2. Cause the victim to *miss* a particular ham email filtered away as *spam*
3. Get a *particular* spam into the victim's inbox
4. Get *any* spam into the victim's inbox

4.2.2 Attacker Knowledge

An attacker may have detailed knowledge of a specific email the victim is likely to receive in the future, or the attacker may know particular words or general information about the victim's word distribution. In many cases, the attacker may know nothing beyond which language the emails are likely to use.

When an attacker wants the victim to see spam emails, a broad dictionary attack can render the spam filter unusable, causing the victim to disable the filter (see Section 4.3.1.1). With more information about the email distribution, the attacker can select a smaller

dictionary of high-value features that are still effective. When an attacker wants to prevent a victim from seeing particular emails and has some information about those emails, the attacker can target them with a *focused attack* (see Section 4.3.1.2). Furthermore, if an attacker can send email messages that the user will train as non-spam, a *pseudospam attack* can cause the filter to accept spam messages into the user's inbox (see Section 4.3.2).

These experimental results confirm that this class of attacks presents a serious concern for statistical spam filters. A dictionary attack makes the spam filter unusable when controlling just 1% of the messages in the training set, and a well-informed focused attack removes the target email from the victim's inbox over 90% of the time. The pseudospam attack causes the victim to see almost 90% of the target spam messages with control of less than 10% of the training data.

I demonstrate the potency of these attacks and present a potential defense—the *Reject On Negative Impact (RONI) defense* tests the impact of each email on training and doesn't train on messages that have a large negative impact. I show that this defense is effective in preventing some attacks from succeeding.

4.2.3 Training Model

SpamBayes produces a *classifier* from a *training set* of labeled examples of spam and non-spam messages. This classifier (or *filter*) is subsequently used to label future email messages as *spam* (bad, unsolicited email) or *ham* (good, legitimate email). SpamBayes also has a third label—when it isn't confident one way or the other, it returns *unsure*. I use the following terminology: the true class of an email can be ham or spam, and a classifier produces the labels *ham*, *spam*, and *unsure*.

There are three natural choices for how to treat *unsure*-labeled messages: they can be placed in the spam folder, they can be left in the user's inbox, or they can be put into a third folder for separate review. Each choice can be problematic because the *unsure* label is likely to appear on both ham and spam messages. If *unsure* messages are placed in the spam folder, the user must sift through all spam periodically or risk missing legitimate messages. If they remain in the inbox, the user will encounter an increased amount of spam messages in their inbox. If they have their own "Unsure" folder, the user still must sift through an increased number of *unsure*-labeled spam messages to locate *unsure*-labeled ham messages. Too much *unsure* email is therefore almost as troublesome as too many false positives (ham labeled as *spam*) or false negatives (spam labeled as *ham*). In the extreme case, if every email is labeled *unsure* then the user must sift through every spam email to find the ham emails and thus obtains no advantage from using the filter.

Consider an organization that uses SpamBayes to filter incoming email for multiple users and periodically retrains on all received email, or an individual who uses SpamBayes as a personal email filter and regularly retrains it with the latest spam and ham. These scenarios serve as canonical usage examples. I use the terms *user* and *victim* interchangeably for either the organization or individual who is the target of the attack; the meaning will be clear from context.

I assume that the user retrains SpamBayes periodically (e.g., weekly); updating the filter in this way is necessary to keep up with changing trends in the statistical characteristics of

both legitimate and spam email. These attacks are not limited to any particular retraining process; they only require the following assumption.

4.2.4 The Contamination Assumption

I assume that the attacker can send emails that the victim will use for training—the *contamination assumption*—but incorporate two significant restrictions: 1) attackers may specify arbitrary email bodies but cannot alter email headers; and 2) attack emails will always be trained as spam, not ham. In the pseudospam attack, however, I investigate the consequences of lifting the second restriction and allowing the attacker to have messages trained as ham.

It is common practice in security research to assume the attacker has as much power as possible, since a determined adversary may find unanticipated methods of attack—if a vulnerability exists, I assume it may be exploited. It is clear that in some cases the attacker can control training data. Here, I discuss realistic scenarios where the contamination assumption is justified; in the later sections, I examine its implications.

Adaptive spam filters must be retrained periodically to cope with the changing nature of both ham and spam. Many users simply train on all email received, using all *spam*-labeled messages as spam training data and all *ham*-labeled messages as ham training data. Generally the user will manually provide true labels for messages labeled *unsure* by the filter, as well as for messages filtered incorrectly as *ham* (false negatives) or *spam*(false positives). In this case, it is trivial for the attacker to control training data: any emails sent to the user are used in training.

The fact that users may manually label emails does not protect against these attacks: the attack messages are unsolicited emails from unknown sources and may contain normal spam marketing content. The *spam* labels manually given to attack emails are correct and yet allow the attack to proceed. When the attack emails can be trained as ham, a different attack is possible; the pseudospam attack explores the case where attack emails are trained as ham (see Section 4.3.2).

4.3 Causative Attacks against SpamBayes' Learner

I present three novel *Causative* attacks against SpamBayes' learning algorithm in the context of the attack taxonomy from Chapter 4.2.1: one is an *Indiscriminate Availability* attack, one is a *Targeted Availability* attack, and the third is a *Targeted Integrity* attack.

These *Causative* attacks against a learning spam filter proceed as follows:

1. The attacker determines the goal for the attack.
2. The attacker sends attack messages to include in the victim's training set.
3. The victim (re-)trains the spam filter, resulting in a contaminated filter.
4. The filter's classification performance degrades on incoming messages.

In the remainder of this section, I describe attacks that achieve the objectives outlined above in Section 4.2. Each of the attacks consists of inserting emails into the training set that are drawn from a particular distribution (*i.e.*, according to the attacker's knowledge as discussed in Section 4.2.2); the properties of these distributions, along with other parameters, determine the nature of the attack. The *dictionary* attack sends email messages with tokens drawn from a broad distribution, essentially including every token with equal probability. The *focused* attack focuses the distribution specifically on one message or a narrow class of messages. If the attacker has the additional ability to send messages that will be trained as ham, a *pseudospam* attack can cause spam messages to reach the user's inbox.

4.3.1 Causative Availability Attacks

I first focus on *Causative Availability* attacks, which manipulate the filter's training data to increase the number of ham messages misclassified. I consider both *Indiscriminate* and *Targeted* attacks. In *Indiscriminate* attacks, enough false positives force the victim to disable the filter or frequently search in *spam/unsure* folders for legitimate messages erroneously filtered away. Hence, the victim is forced to view more spam. In *Targeted* attacks, the attacker does not disable the filter but surreptitiously prevents the victim from receiving certain messages.

Without loss of generality, consider the construction of a single attack message A. The victim adds it to the training set, (re-)trains on the contaminated data, and subsequently uses the tainted model to classify a new message $\hat{X}$. The attacker also has some (perhaps limited) knowledge of the next email the victim will receive. This knowledge can be represented as a distribution $\mathbf{p}$—the vector of probabilities that each token will appear in the next message.

The goal of the attacker is to choose the tokens for the attack message $\mathbf{a}$ to maximize the *expected spam score:*

$$\max_{\mathbf{a}} \mathrm{E}_{\hat{\mathbf{x}}\sim\mathbf{p}} \left[I_{\mathbf{a}} \left(\hat{\mathbf{x}} \right) \right] \quad ; \tag{4.7}$$

that is, the attack goal is to maximize the expectation of $I_{\mathbf{a}}(\hat{\mathbf{x}})$ (Equation (4.4) with the attack message $\mathbf{a}$ added to the spam training set) of the next legitimate email $\hat{\mathbf{x}}$ drawn from distribution $\mathbf{p}$. However, in analyzing this objective, it is shown in Appendix B.2 that the attacker can generally maximize the expected spam score of any future message by including *all possible tokens* (words, symbols, misspellings, etc.) in attack emails, causing SpamBayes to learn that all tokens are indicative of spam—I call this an *Optimal* attack[3].

To describe the optimal attack under this criterion, I make two observations, which I detail in Appendix B.2. First, for most tokens, $I_{\mathbf{a}}(\cdot)$ is monotonically non-decreasing in q_i. Therefore, increasing the score of any token in the attack message will generally increase $I_{\mathbf{a}}(\hat{\mathbf{x}})$. Second, the token scores of distinct tokens do not interact; that is, adding the i^{th} token to the attack does not change the score q_j of some different token $j \neq i$. Hence, the attacker can simply choose which tokens will be most beneficial for their purpose. From

[3] As discussed in Appendix B.2 these attacks are optimal for a relaxed version of the optimization problem. Generally, optimizing the problem given by Equation 4.7 requires exact knowledge about future messages $\hat{\mathbf{x}}$ and is a difficult combinatorial problem to solve.

this, I motivate two attacks, the *dictionary* and *focused* attacks, as instances of a common attack in which the attacker has different amounts of knowledge about the victim's email.

For this, let us consider specific choices for the distribution $\mathbf{p}$. First, if the attacker has little knowledge about the tokens in target emails, we give equal probability to each token in $\mathbf{p}$. In this case, one can optimize the expected message spam score by including *all possible tokens* in the attack email. Second, if the attacker has specific knowledge of a target email, we can represent this by setting $\mathbf{p}_i$ to 1 if and only if the i^{th} token is in the target email. This attack is also optimal with respect to the target message, but it is much more compact.

In practice, the optimal attack requires intractably large attack messages, but the attacker can exploit his knowledge about the victim (captured by $\mathbf{p}$) to approximate the effect of an optimal attack by instead using a large set of common words that the victim is likely to use in the future such as a dictionary—hence these are *dictionary attacks*. If the attacker has relatively little knowledge, such as knowledge that the victim's primary language is English, the attack can include all words in an English dictionary. This reasoning yields the *dictionary attack* (see Section 4.3.1.1). On the other hand, the attacker may know *some* of the particular words to appear in a target email, though not all of the words. This scenario is the *focused attack* (see Section 4.3.1.2). Between these levels of knowledge, an attacker could use information about the distribution of words in English text to make the attack more efficient, such as characteristic vocabulary or jargon typical of emails the victim receives. Any of these cases result in a distribution $\mathbf{p}$ over tokens in the victim's email that is more specific than an equal distribution over all tokens but less informative than the true distribution of tokens in the next message. Below, I explore the details of the dictionary and focused attacks, with some exploration of using an additional corpus of common tokens to improve the dictionary attack.

4.3.1.1 Dictionary Attack

The *dictionary* attack, an *Indiscriminate* attack, makes the spam filter unusable by causing it to misclassify a significant portion of ham emails (*i.e.*, causing false positives) so that the victim loses confidence in his filter. As a consequence either the victim disables his spam filter, or at least must frequently search through *spam/unsure* folders to find legitimate messages that were incorrectly classified. In either case, the victim loses confidence in the filter and is forced to view more spam achieving the ultimate goal of the spammer: the victim views desired spams while searching for legitimate mail. The result of this attack is denial of service; *i.e.*, a higher rate of ham misclassified as *spam*.

The dictionary attack is an approximation of the optimal attack suggested in Section 4.3.1, in which the attacker maximizes the expected score by including all possible tokens. Creating messages with every possible token is infeasible in practice. Nevertheless, when the attacker lacks knowledge about the victim's email, this optimal attack can be approximated by the set of all tokens that the victim is likely to use such as a dictionary of the victim's native language—I call this a *dictionary attack*. The dictionary attack increases the score of every token in a dictionary; *i.e.*, it makes them more indicative of spam.

The central idea that underlies the dictionary attack is to send attack messages containing a large set of tokens—the attacker's *dictionary*. The dictionary is selected as the set of

tokens whose scores maximally increase the expected value of $I_{\mathbf{a}}(\hat{\mathbf{x}})$ as in Equation (4.7). Since the score of a token typically increases when included in an attack message (except in unusual circumstances as described in Appendix B), the attacker can simply include any tokens that are likely to occur in future legitimate messages according to the attacker's knowledge from the distribution $\mathbf{p}$. In particular, if the victim's language is known by the attacker, he can use that language's entire lexicon (or at least a large subset) as the attack dictionary. After training on a set of dictionary messages, the victim's spam filter will have a higher spam score for every token in the dictionary, an effect which is amplified for rare tokens. As a result, future legitimate email is more likely to be marked as *spam* since it will contain many tokens from that lexicon.

A refinement of this attack instead uses a token source with a distribution closer to the victim's true email distribution. For example, a large pool of *Usenet* newsgroup postings may have colloquialisms, misspellings, and other words not found in a proper dictionary. Furthermore, using the most frequent tokens in such a corpus may allow the attacker to send smaller emails without losing much effectiveness. However, there is an inherent trade-off in choosing tokens. Rare tokens are the most vulnerable to attack since their scores will shift more towards spam (a spam score of 1.0 given by the score in Equation (4.4)) with fewer attack emails. However, the rare vulnerable tokens also are less likely to appear in future messages, diluting their usefulness.

In my experiments (Section 4.5.2), I evaluate two variants of the dictionary attacks: the first is based on the *Aspell* dictionary and the second on a dictionary compiled from the most common tokens observed in a *Usenet* corpus. I refer to these as the *Aspell* and *Usenet* dictionary attacks respectively.

4.3.1.2 Focused Attack

The second *Causative Availability* attack is a *Targeted* attack—the attacker has some knowledge of a specific legitimate email he targets to be incorrectly filtered. If the attacker has exact knowledge of the target email, placing all of its tokens in attack emails produces an optimal targeted attack. Realistically, though, the attacker only has partial knowledge about the target email and can guess only some of its tokens to include in attack emails. I model this knowledge by letting the attacker know a certain fraction of tokens from the target email, which are included in the attack message. The attacker constructs attack email that contain words likely to occur in the target email; *i.e.*, the tokens known by the attacker. The attack email may also include additional tokens added by the attacker to obfuscate the attack message's intent since extraneous tokens do not impact the attack's effect on the targeted tokens. When SpamBayes trains on the resulting attack email, the spam scores of the targeted tokens generally increase (see Appendix B), so the target message is more likely to be filtered as *spam*. This is the focused attack.

For example, an unscrupulous company may wish to prevent its competitors from receiving email about a competitive bidding process and they know specific words that will appear in the target email, obviating the need to include the entire dictionary in their attacks. They attack by sending spam emails to the victim with tokens such as the names of competing companies, their products, and their employees. Further, if the bid messages follow a common template known to the malicious company, this further facilitates their at-

tack. As a result of the attack, legitimate bid emails may be filtered away as *spam*, causing the victim not to see it.

The focused attack is more concise than the dictionary attack because the attacker has detailed knowledge of the target email and no reason to affect other messages. This conciseness makes the attack both more efficient for the attacker and more difficult to detect for the defender. Further, the focused attack can be more effective because the attacker may know proper nouns and other non-word tokens common in the victim's email that are otherwise uncommon in typical English text.

An interesting side-effect of the focused attack is that repeatedly sending similar emails tends to not only increase the spam score of tokens in the attack but also *reduce* the spam score of tokens not in the attack. To understand why, recall the estimate of the token posterior in Equation (4.1), and suppose that the j^{th} token does not occur in the attack email. Then $N^{(s)}$ increases with the addition of the attack email but $n_j^{(s)}$ does not, so $P_j^{(S)}$ decreases and therefore so does q_j. In Section 4.5.3, I observe empirically that the focused attack can indeed reduce the spam score of tokens not included in the attack emails.

4.3.2 Causative Integrity Attacks—Pseudospam

I also study *Causative Integrity* attacks, which manipulate the filter's training data to increase false negatives; that is, spam messages misclassified as *ham*. In contrast to the previous attacks, the *pseudospam attack* directly attempts to make the filter misclassify spam messages. If the attacker can choose messages arbitrarily that are trained as ham, the attack is similar to a focused attack with knowledge of 100% of the target email's tokens. However, there is no reason to believe a user would train on arbitrary messages as ham. I introduce the concept of a *pseudospam email*—an email that does not look like spam but that has characteristics (such as headers) that are typical of true spam emails. Not all users consider benign-looking, non-commercial emails offensive enough to mark them as spam.

To create pseudospam emails, I take the message body text from newspaper articles, journals, books, or a corpus of legitimate email. The idea is that in some cases, users may mistake these messages as ham for training, or may not be diligent about correcting false negatives before retraining, if the messages do not have marketing content. In this way, an attacker might be able to gain control of ham training data. This motivation is less compelling than the motivation for the dictionary and focused attacks, but in the cases where it applies, the headers in the pseudospam messages will gain significant weight indicating ham, so when future spam is sent with similar headers (*i.e.*, by the same spammer) it will arrive in the user's inbox.

4.4 The Reject On Negative Impact (RONI) defense

In his Master's thesis, Udam Saini studied two defense strategies for countering *Causative Availability* attacks on SpamBayes [Saini, 2008]. The first was a mechanism to adapt SpamBayes' threshold parameters to mitigate the impact of an *Availability* attack called the threshold defense. This defense did reduce the false positive rate of *dictionary* but at

a cost of a higher false negative rate. He also discussed a preliminary version of the RONI defense, which I elaborate on here.

In Chapter 3.5.4.1, I summarized the Reject On Negative Impact (RONI) defense. As stated in that section, the RONI defense measures the empirical effect of each training instance and eliminates from training those points that have a substantial negative impact on classification accuracy. To determine whether a candidate training instance is malicious or not, the defender trains a classifier on a base training set, then adds the candidate instance to his training set and trains a second classifier with the candidate included. The defender applies both classifiers to a *quiz set* of instances with known labels, measuring the difference in accuracy between the two. If adding the candidate instance to the training set causes the resulting classifier to produce substantially more classification errors, the instance is rejected from the training set due to its detrimental effect.

More formally, I assume there is an initial training set $\mathbb{D}^{(\text{train})}$ and a set $\mathbb{D}^{(\text{suspect})}$ of additional candidate training points to be added to the training set. The points in $\mathbb{D}^{(\text{suspect})}$ are assessed as follows: first a *calibration set* $\mathbb{C}$, which is a randomly chosen subset of $\mathbb{D}^{(\text{train})}$, is set aside. Then several independent and potentially overlapping training/quiz set pairs $\langle \mathbb{T}_i, \mathbb{Q}_i \rangle$ are sampled from the remaining portion of $\mathbb{D}^{(\text{train})}$, where the points within a pair of sets are sampled without replacement. To assess the impact (empirical effect) of a data point $\langle x, y \rangle \in \mathbb{D}^{(\text{suspect})}$, for each pair of sets $(\mathbb{T}_i, \mathbb{Q}_i)$ one constructs a *before* classifier f_i trained on $\mathbb{T}_i$ and an *after* classifier $\hat{f}_i$ trained on $\mathbb{T}_i + \langle x, y \rangle$; *i.e.*, the sampled training set with $\langle x, y \rangle$ concatenated. The RONI defense then compares the classification accuracy of f_i and $\hat{f}_i$ on the quiz set $\mathbb{Q}_i$, using the change in true positives and true negatives caused by adding $\langle x, y \rangle$ to $\mathbb{T}_i$. If either change is significantly negative when averaged over training/quiz set pairs, $\langle x, y \rangle$ is considered to be too detrimental, and it is excluded from $\mathbb{D}^{(\text{train})}$. To determine the significance of a change, the shift in accuracy of the detector is compared to the average shift caused by points in the calibration set $\mathbb{C}$. Each point in $\mathbb{C}$ is evaluated in a way analogous to evaluation of the points in $\mathbb{D}^{(\text{suspect})}$. The median and standard deviation of their true positive and true negative changes is computed, and the significance threshold is chosen to be the third standard deviation below the median.

4.5 Experiments with SpamBayes

4.5.1 Experimental Method

Here I present an empirical evaluation of the impact of *Causative Availability* attacks on SpamBayes' spam classification accuracy.

4.5.1.1 Datasets

In these experiments, I use the Text Retrieval Conference (TREC) 2005 spam corpus as described by Cormack and Lynam [2005], which is based on the Enron email corpus [Klimt and Yang, 2004] and contains 92,189 emails (52,790 spam and 39,399 ham). By sampling from this dataset, I construct sample inboxes and measure the effect of injecting attacks into them. This corpus has several strengths: it comes from a real-world source, it has

a large number of emails, and its creators took care that the added spam does not have obvious artifacts to differentiate it from the ham.

I use two sources of tokens for attacks. First, I use the GNU Aspell English dictionary version 6.0-0, containing 98,568 words. I also use a corpus of English Usenet postings to generate tokens for the attacks. This corpus is a subset of a Usenet corpus of 140,179 postings compiled by the University of Alberta's Westbury Lab [Shaoul and Westbury, 2007]. An attacker can download such data and build a language model to use in attacks, and I explore how effective this technique is. I build a primary Usenet dictionary by taking the most frequent 90,000 tokens in the corpus (Usenet-90k), and I also experiment with a smaller dictionary of the most frequent 25,000 tokens (Usenet-25k).

The overlap between the Aspell dictionary and the most frequent 90,000 tokens in the Usenet corpus is approximately 26,800 tokens. The overlap between the Aspell dictionary and the TREC corpus is about 16,100 tokens, and the intersection of the TREC corpus and Usenet-90k is around 26,600 tokens.

4.5.1.2 Constructing Message Sets for Experiments

In constructing an experiment, I often need several non-repeating sequences of emails in the form of mailboxes. When I require a mailbox, I sample messages without replacement from the TREC corpus, stratifying the sampling to ensure the necessary proportions of ham and spam. For subsequent messages needed in any part of the experiment (target messages, headers for attack messages, and so on), I again sample emails without replacement from the messages remaining in the TREC corpus. In this way, I ensure that no message is repeated within the experiment.

I construct attack messages by splicing elements of several emails together to make messages that are realistic under a particular model of the adversary's control. I construct the attack email bodies according to the specifications of the attack. I select the header for each attack email by choosing a random spam email from TREC and using its headers, taking care to ensure that the content-type and other Multipurpose Internet Mail Extensions (MIME) headers correctly reflect the composition of the attack message body. Specifically, I discard the entire existing multi- or single-part body and I set relevant headers (such as Content-Type and Content-Transfer-Encoding) to indicate a single plain-text body.

The tokens used in each attack message are selected from the datasets according to the attack method. For the dictionary attack, I use all tokens from the attack dictionary in every attack message (98,568 tokens for the Aspell dictionary and 90,000 or 25,000 tokens for the Usenet dictionary). For the focused and the pseudospam attacks, I select tokens for each attack message based on a fresh message sampled from the TREC dataset. The number of tokens in attack messages for the focused and pseudospam attacks varies, but all such messages are comparable in size to the messages in the TREC dataset.

Finally, to evaluate an attack, I create a control model by training SpamBayes once on the base training set. I incrementally add attack emails to the training set and train new models at each step, yielding a sequence of models tainted with increasing numbers of attack messages. (Because SpamBayes is order-independent in its training, it arrives at the same model whether training on all messages in one batch or training incrementally on

Parameter	Focused Attack	PseudoSpam Attack	RONI Defense
Training set size	2,000, 10,000	2,000, 10,000	2,000, 10,000
Test set size	200, 1,000	200, 1,000	N/A
Spam prevalence	0.50, 0.75, 0.90	0.50, 0.75, 0.90	0.50
Attack fraction	0.001, 0.005, 0.01, 0.02, 0.05, 0.10	0.001, 0.005, 0.01, 0.02, 0.05, 0.10	0.10
Folds of validation	10	10	N/A
Target Emails	20	N/A	N/A

Table 4.1: Parameters used in the experiments on attacking SpamBayes.

each email in any order.) I evaluate the performance of these models on a fresh set of test messages.

4.5.1.3 Attack Assessment Method

I measure the effect of each attack by randomly choosing an inbox according to the parameters in Table 4.1 and comparing classification performance of the control and compromised filters using ten-fold cross-validation. In cross-validation, I partition the data into ten subsets and perform ten train-test epochs. During the k^{th} epoch, the k^{th} subset is set aside as a test set and the remaining subsets are combined into a training set. In this way, each email from the sample inbox functions independently as both training and test data.

In the sequel, I demonstrate the effectiveness of attacks on test sets of held-out messages. Because the dictionary and focused attacks are designed to cause ham to be misclassified, I only show their effect on ham messages; I found that their effect on spam is marginal. Likewise, for the pseudospam attack, I concentrate on the results for spam messages. Most of my graphs do not include error bars since I observed that the variation in the tests was small compared to the effect of the attacks (see Figure (b) and (d)). See Table 4.1 for the parameters used in the experiments. I found that varying the size of the training set and spam prevalence in the training set had minimal impact on the performance of the attacks (for comparison, see Figure (a) and (c)), so I primarily present the results of $10,000$-message training sets at 50% spam prevalence.

4.5.2 Dictionary Attack Results

I examine dictionary attacks as a function of the percent of attack messages in the training set. Figures 4.2 show the misclassification rates of three dictionary attack variants averaging over ten-fold cross-validation in two settings (Figures (a) and (b) have an initial training set of $10,000$ messages with 50% spam while Figures (c) and (d) have an initial training set of $2,000$ messages with 75% spam). First, I analyze the optimal dictionary attack discussed in Section 4.3.1 by simulating the effect of including every possible token in our attack emails. As shown in the figures, this optimal attack quickly causes the filter to mislabel all ham emails with only a minute fraction of control of the training set.

Dictionary attacks using tokens from the Aspell dictionary are also successful, though not as successful as the optimal attack. Both the Usenet-90k and Usenet-25k dictionary attacks cause more ham emails to be misclassified than the Aspell dictionary attack, since

they contains common misspellings and slang terms that are not present in the Aspell dictionary. All of these variations of the attack require relatively few attack emails to significantly degrade SpamBayes' accuracy. After 101 attack emails (1% of 10,000), the accuracy of the filter falls significantly for each attack variation. Overall misclassification rates are 96% for optimal, 37% for Usenet-90k, 19% for Usenet-25k, and 18% for Aspell—at this point most users will gain no advantage from continued use of the filter so the attack has succeeded.

It is of significant interest that so few attack messages can degrade a common filtering algorithm to such a degree. However, while the attack emails make up a small percentage of the *number of messages* in a contaminated inbox, they make up a large percentage of the *number of tokens*. For example, at 204 attack emails (2% of the training messages), the Usenet-25k attack uses approximately 1.8 times as many tokens as the entire pre-attack training dataset, and the Aspell attack includes 7 times as many tokens.

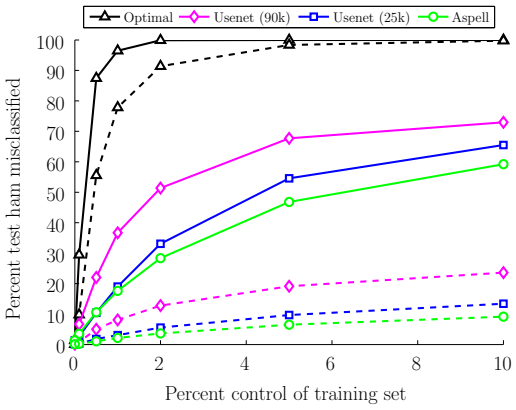
While it seems trivial to prevent dictionary attacks by filtering large messages out of the training set, such strategies fail to completely address this vulnerability of SpamBayes. First, while ham messages in TREC are relatively small (fewer than 1% exceeded 5,000 tokens and fewer than 0.01% of messages exceeded 25,000 tokens), this dataset has been redacted to remove many attachments and hence may not be representative of actual messages. Second, an attacker can circumvent size-based thresholds. By fragmenting the dictionary, an attack can have a similar impact using more messages with fewer tokens per message. Additionally, informed token selection methods can yield more effective dictionaries as I demonstrate with the two Usenet dictionaries. Thus, size-based defenses lead to a trade-off between vulnerability to dictionary attacks and the effectiveness of training the filter. In the next section, I present a defense that instead filters messages based directly on their impact on the spam filter's accuracy.

4.5.3 Focused Attack Results

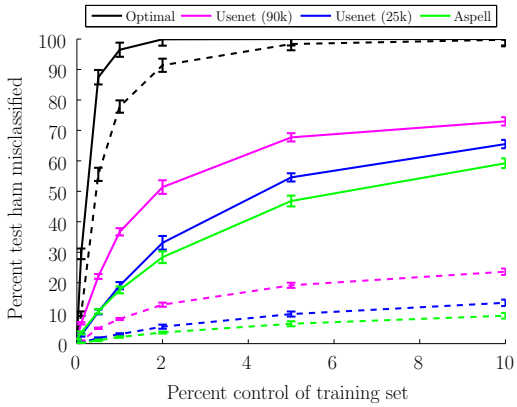
In this section, I discuss experiments examining how accurate the attacker needs to be at guessing target tokens, how many attack emails are required for the focused attack to be effective, and what effect the focused attack has on the token scores of a targeted message. For the focused attack, I randomly select 20 ham emails from the TREC corpus to serve as the target emails before creating the clean training set. During each fold of cross-validation, I executed 20 focused attacks, one for each email, so the results average over 200 different trials.

These results differ from the focused attack experiments conducted in Nelson et al. [2008] in two important ways. First, here I randomly select a fixed percentage of tokens known by the attacker from each message instead of selecting each token with a fixed probability. The later approach causes the percentage of tokens known by the attacker to fluctuate from message to message. Second, I only select messages with more than 100 tokens to use as target emails. With these changes, these results more accurately represent the behavior of a focused attack. Furthermore, in this more accurate setting, the focused attack is even more effective.

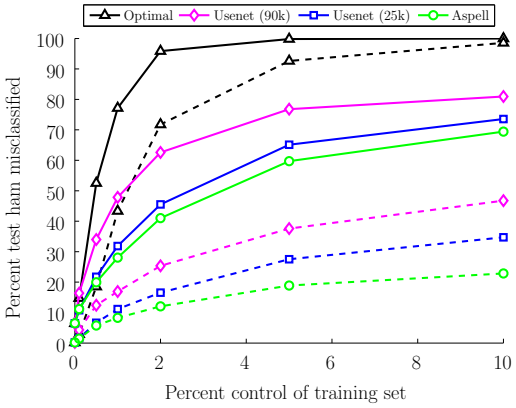
Figure 4.3 shows the effectiveness of the attack when the attacker has increasing knowledge of the target email by simulating the process of the attacker guessing tokens from the



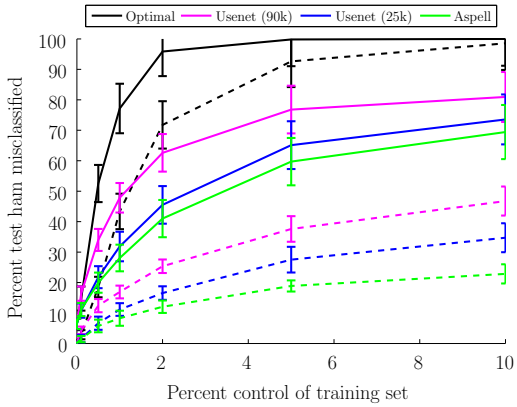
(a) Training on 10,000 messages (50% spam)



(b) Attacks (with error bars)



(c) Training on 2,000 messages (75% spam)



(d) Attacks (with error bars)

Figure 4.2: Effect of three dictionary attacks on SpamBayes in two settings. Figure **(a)** and **(b)** have an initial training set of 10,000 messages (50% spam) while Figure **(c)** and **(d)** have an initial training set of 2,000 messages (75% spam). Figure **(b)** and **(d)** also depict the standard errors in the experiments for both of the settings. I plot percent of ham classified as *spam* (dashed lines) and as *spam* or *unsure* (solid lines) against the attack as percent of the training set. I show the optimal attack ($\triangle$), the Usenet-90k dictionary attack ($\Diamond$), the Usenet-25k dictionary attack ($\square$), and the Aspell dictionary attack ($\bigcirc$). Each attack renders the filter unusable with adversarial control over as little as 1% of the messages (101 messages).

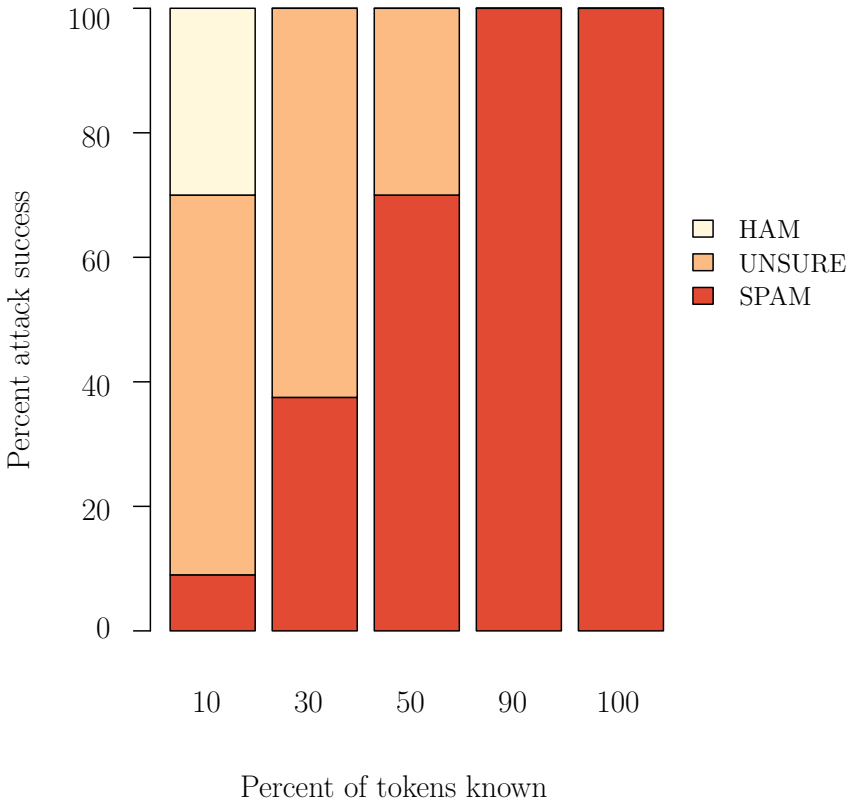


Figure 4.3: Effect of the focused attack as a function of the percentage of target tokens known by the attacker. Each bar depicts the fraction of target emails classified as *spam*, *ham*, and *unsure* after the attack. The initial inbox contains 10,000 emails (50% spam).

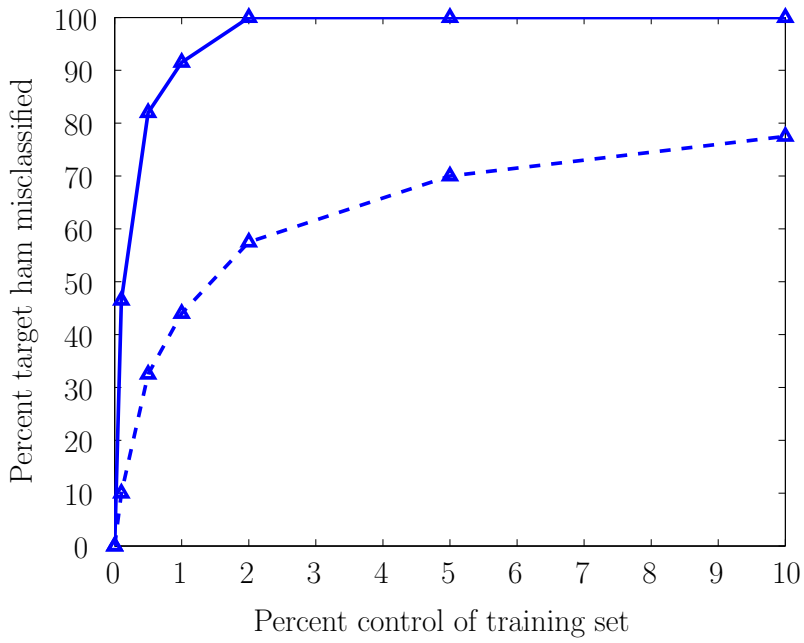


Figure 4.4: Effect of the focused attack as a function of the number of attack emails with a fixed fraction (F=0.5) of tokens known by the attacker. The dashed line shows the percentage of target ham messages classified as *spam* after the attack, and the solid line the percentage of targets that are *spam* or *unsure* after the attack. The initial inbox contains 10,000 emails (50% spam).

target email. I assume that the attacker knows a fixed fraction F of the actual tokens in the target email, with $F \in \{0.1, 0.3, 0.5, 0.9\}$—the x-axis of Figure 4.3. The y-axis shows the percent of the 20 targets classified as *ham*, *unsure* and *spam*. As expected, the attack is increasingly effective as F increases. If the attacker knows 50% of the tokens in the target, classification changes to *spam* or *unsure* on *all* of the target emails, with a 75% rate of classifying as *spam*.

Figure 4.4 shows the attack's effect on misclassifications of the target emails as the number of attack messages increases with the fraction of known tokens fixed at 50%. The x-axis shows the number of messages in the attack as a fraction of the training set, and the y-axis shows the fraction of target messages misclassified. With 101 attack emails inserted into an initial mailbox size of $10,000$ (1%), the target email is misclassified as *spam* or *unsure* over 90% of the time.

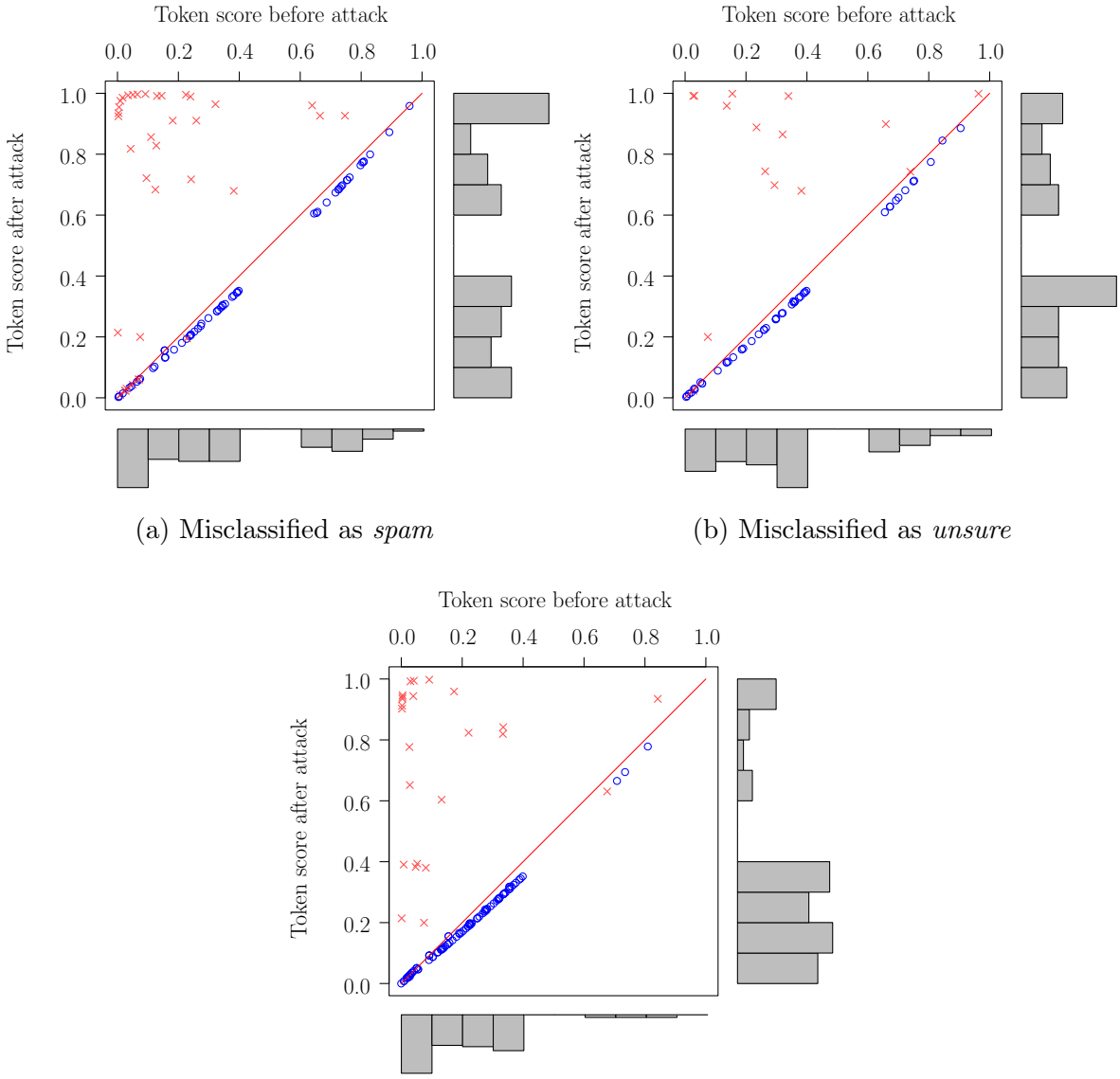
Figure 4.5 shows the attack's effect on three representative emails. Each of the graphs in the figure represents a single target email from each of three attack results: ham misclassified as *spam* (Figure (a)), ham misclassified as *unsure* (Figure (b)), and ham correctly classified as *ham* (Figure (c)). Each point represents a token in the email. The x-axis is the token's spam score (from Equation (4.2)) before the attack, and the y-axis is the token's score after the attack (0 indicates *ham* and 1 indicates *spam*). The $\times$'s are tokens included in the attack (known by the attacker) and the $\bigcirc$'s are tokens not in the attack. The histograms show the distribution of token scores before the attack (at bottom) and after the attack (at right).

Any point above the line $y = x$ is a token whose score increased due to the attack and any point below is a decrease. These graphs demonstrate that the score of the tokens included in the attack typically increase significantly while those not included decrease slightly. Since the increase in score is more significant for included tokens than the decrease in score for excluded tokens, the attack has substantial impact even when the attacker has a low probability of guessing tokens, as seen in Figure 4.3. Further, the before/after histograms in Figure 4.5 provide a direct indication of the attack's success. In shifting most token scores toward 1, the attack causes more misclassifications.

4.5.4 Pseudospam Attack Experiments

In contrast to the previous attacks, for the pseudospam attack, I created attack emails that may be labeled as ham by a human as the emails are added into the training set. I setup the experiment for the pseudospam attack by first randomly selecting a target spam header to be used as the base header for the attack. I then create the set of attack emails that look similar to ham emails (see Section 4.3.2). To create attack messages, I combine each ham email with the target spam header. This is done so that the attack email has contents similar to other legitimate email messages. Header fields that may modify the interpretation of the body are taken from the ham email to make the attack realistic.

Figure 4.6 demonstrates the effectiveness of the *pseudospam attack* by plotting the percent of attack messages in the training set (x-axis) against the misclassification rates on the test spam email (y-axis). The solid line shows the fraction of target spam classified as *ham* or *unsure* spam while the dashed line shows the fraction of spam classified as *ham*. In the absence of attack, SpamBayes only misclassifies about 10% of the target spam emails



(a) Misclassified as *spam*

(b) Misclassified as *unsure*

(c) Correctly classified as *ham*

Figure 4.5: Effect of the focused attack on three representative emails—one graph for each target. Each point is a token in the email. The x-axis is the token's spam score in Equation (4.2) before the attack (0 indicates *ham* and 1 indicates *spam*). The y-axis is the token's spam score after the attack. The $\times$'s are tokens that were included in the attack and the $\bigcirc$'s are tokens that were not in the attack. The histograms show the distribution of spam scores before the attack (at bottom) and after the attack (at right).

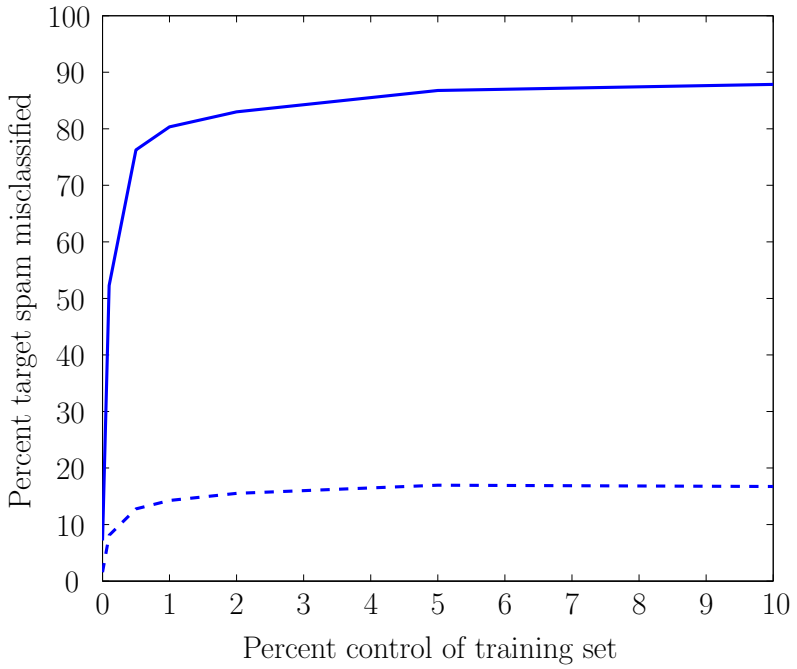


Figure 4.6: Effect of the pseudospam attack when trained as ham as a function of the number of attack emails. The dashed line shows the percentage of the adversary's messages classified as *ham* after the attack, and the solid line the percentage that are *ham* or *unsure* after the attack. The initial inbox contains 10,000 emails (50% spam).

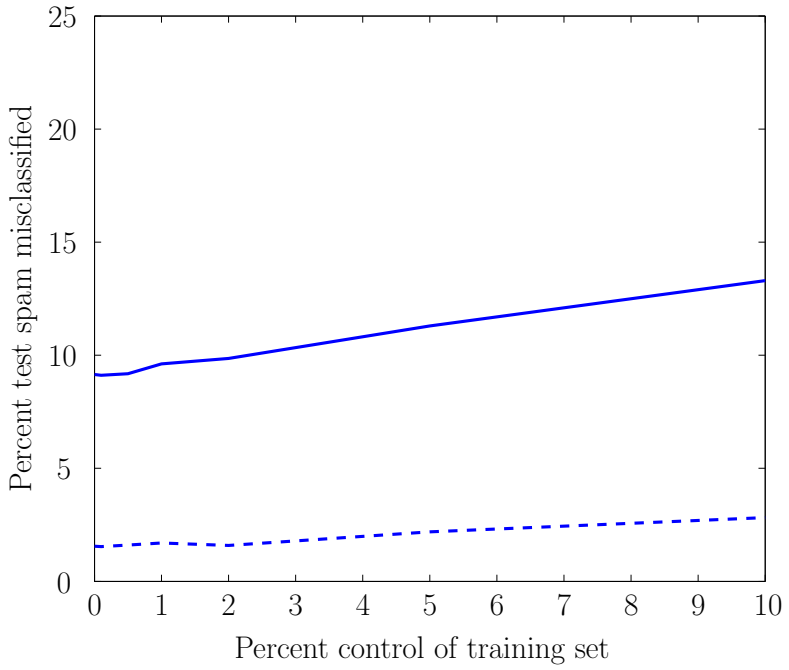


Figure 4.7: Effect of the pseudospam attack when trained as spam, as a function of the number of attack emails. The dashed line shows the percentage of the normal spam messages classified as *ham* after the attack, and the solid line the percentage that are *unsure* after the attack. Surprisingly, training the attack emails as ham causes an increase in misclassification of normal spam messages. The initial inbox contains 10,000 emails (50% spam).

(including those labeled *unsure*). If the attacker can insert a few hundred attack emails (1% of the training set), then SpamBayes misclassifies more than 80% of the target spam emails.

Further, the attack has a minimal effect on regular ham and spam messages. Other spam email messages are still correctly classified since they do not generally have the same header fields as the adversary's messages. In fact, ham messages may have lower spam scores since they may contain tokens similar to those in the attack emails.

I also explore the scenario in which the pseudospam attack emails are labeled by the user as *spam* to better understand the effect of these attacks if the pseudospam messages fail to fool the user. The result is that, in general, SpamBayes classifies more spam messages incorrectly. As Figure 4.7 indicates, this variant causes an increase in spams mislabeled as either *unsure* or *ham* increases to nearly 15% as the number of attack emails increases. Further, this version of the attack does not cause a substantial impact on normal ham messages.

4.5.5 RONI defense Results

Again to empirically evaluate the RONI defense, I sample inboxes from the TREC 2005 spam corpus. In this assessment, I use 20-fold cross validation to get an initial training inbox $\mathbb{D}^{(\text{train})}$ of about $1,000$ messages (50% spam) and a test set $\mathbb{D}^{(\text{eval})}$ of about 50 messages. I also sample a separate set $\mathbb{D}^{(\text{suspect})}$ of $1,000$ additional messages from the TREC corpus to test as a baseline. In each fold of cross validation, I run five separate trials of the RONI defense. For each trial, I use a calibration set of 25 ham and 25 spam messages and sample three training/quiz set pairs of 100 training and 100 quiz messages from the remaining 950 messages. I train two classifiers on each training set for each message in $\mathbb{D}^{(\text{suspect})}$, one with and one without the message, measuring performance on the corresponding quiz set and comparing it to the magnitude of change measured from the calibration set.

I perform the RONI defense evaluation for each message in $\mathbb{D}^{(\text{suspect})}$ as just described to see the effect on non-attack emails. I find that the RONI defense (incorrectly) rejects an average of 2.8% of the ham and 3.1% of the spam from $\mathbb{D}^{(\text{suspect})}$. To evaluate the performance of the post-RONI defense filter, I train a classifier on all messages in $\mathbb{D}^{(\text{suspect})}$ and a second classifier on the messages in $\mathbb{D}^{(\text{suspect})}$ not rejected by the RONI defense. When trained on all $1,000$ messages, the resulting filter correctly classifies 98% of ham and 80% of the spam. After removing the messages rejected by the RONI defense and training from scratch, the resulting filter still correctly classifies 95% of ham and 87% of the spam. The overall effect of the RONI defense on classification accuracy is shown in Figure 4.2.

Since the RONI defense removes non-attack emails in this test, and therefore removing potentially useful information from the training data, SpamBayes' classification accuracy suffers. It is interesting to see that test performance on spam actually improves after removing some emails from the training set. This result seems to indicate that some non-attack emails confuse the filter more than they help when used in training, perhaps because they happen to naturally fit some of the characteristics that attackers use in emails.

Next I evaluate the performance of the RONI defense where $\mathbb{D}^{(\text{suspect})}$ instead consists of attack emails from the attacks described earlier in Sections 4.3. The RONI defense rejects every single dictionary attack from any of the dictionaries (optimal, Aspell, and Usenet). In fact, the degree of change in misclassification rates for each dictionary message is greater than five standard deviations from the median, suggesting that these attacks are easily eliminated with only minor impact on the performance of the filter. See Figure 4.3.

A similar experiment with attack emails from the focused attack shows that the RONI defense is much less effective against focused attack messages. The likely explanation is simple: *Indiscriminate* dictionary attacks broadly affect many different messages with their wide scope of tokens, so its consequences are likely to be seen in the quiz sets. The focused attack is instead targeted at a single *future* email, which may not bear any significant similarity to the messages in the quiz sets. However, as the fraction of tokens correctly guessed by the attacker increases, the RONI defense identifies increasingly many attack messages: only 7% are removed when the attacker guesses 10% of the tokens but 25% of the attacks are removed when the attacker guesses 100% of the tokens. This is likely due to the fact that with more correctly guessed tokens, the overlap with other messages increases sufficiently to trigger the RONI defense more frequently. However, the attack is still successful in spite of the increased number of detections. See Figure 4.4.

Before the RONI defense

		Predicted Label		
		ham	spam	unsure
Truth	ham	97%	0.0%	2.5%
	spam	2.6%	80%	18%

After the RONI defense

		Predicted Label		
		ham	spam	unsure
Truth	ham	95%	0.3%	4.6%
	spam	2.0%	87%	11%

Table 4.2: Effect of the RONI defense on the accuracy of SpamBayes in the absence of attacks. Each confusion matrix shows the breakdown of SpamBayes's predicted labels for both ham and spam messages. **Left:** The average performance of SpamBayes on training inboxes of about 1,000 message (50% spam). **Right:** The average performance of SpamBayes after the training inbox is censored using the RONI defense. On average, the RONI defense removes 2.8% of ham and 3.1% of spam from the training sets. (Numbers may not add up to 100% because of rounding error.)

Dictionary Attacks (Before the RONI defense)

		Predicted Label		
		ham	spam	unsure
Optimal				
True Label	ham	4.6%	83%	12%
	spam	0.0%	100%	0.0%
Aspell				
True Label	ham	66%	12%	23%
	spam	0.0%	98%	1.6%
Usenet				
True Label	ham	47%	24%	29%
	spam	0.0%	99%	0.9%

Dictionary Attacks (After the RONI defense)

		Predicted Label		
		ham	spam	unsure
Optimal				
True Label	ham	95%	0.3%	4.6%
	spam	2.0%	87%	11%
Aspell				
True Label	ham	95%	0.3%	4.6%
	spam	2.0%	87%	11%
Usenet				
True Label	ham	95%	0.3%	4.6%
	spam	2.0%	87%	11%

Table 4.3: I apply the RONI defense to dictionary attacks with 1% contamination of training inboxes of about 1,000 messages (50% spam) each. **Left:** The average effect of optimal, Usenet, and Aspell attacks on the SpamBayes filter's classification accuracy. The confusion matrix shows the breakdown of SpamBayes's predicted labels for both ham and spam messages after the filter is contaminated by each dictionary attack. **Right:** The average effect of the dictionary attacks on their targets after application of the RONI defense. By using the RONI defense, all of these dictionary attacks are caught and removed from the training set, which dramatically improves the accuracy of the filter.

Focused Attacks (Before the RONI defense)

	Target Prediction		
	ham	spam	unsure
10% guessed	78%	0.0%	22%
30% guessed	30%	5.2%	65%
50% guessed	5.8%	23%	71%
90% guessed	0.0%	79%	21%
100% guessed	0.0%	86%	14%

Focused Attacks (After the RONI defense)

	Target Prediction		
	ham	spam	unsure
10% guessed	79%	2.7%	21%
30% guessed	36%	4.8%	59%
50% guessed	19%	20%	61%
90% guessed	20%	62%	19%
100% guessed	21%	66%	13%

Table 4.4: The RONI defense to focused attacks with 1% contamination of training inboxes of about $1,000$ messages (50% spam) each. **Left:** The average effect of 35 focused attacks on their targets when the attacker correctly guesses 10, 30, 50, 90, and 100% of the target's tokens. **Right:** The average effect of the focused attacks on their targets after application of the RONI defense. By using the RONI defense, more of the target messages are correctly classified as ham, but the focused attacks largely still succeed at misclassifying most targeted messages.

4.6 Summary

Motivated by the taxonomy of attacks against learners, I designed real-world *Causative* attacks against SpamBayes' learner and demonstrated the effectiveness of these attacks using realistic adversarial control over the training process of SpamBayes. Optimal attacks against SpamBayes caused unusably high false positive rates using only a small amount of control of the training process (more than 95% misclassification of ham messages when only 1% of the training data is contaminated). Usenet dictionary attack also effectively use a more realistically limited attack message to cause misclassification of 19% of ham messages with only 1% control over the training messages, rendering SpamBayes unusable in practice. I also show that an informed adversary can successfully target messages. The focused attack changes the classification of the target message virtually 100% of the time with knowledge of only 30% of the target's tokens. Similarly, the pseudospam attack is able to cause nearly 90% of the target spam messages to be labeled as either *unsure* or *ham* with control of less than 10% of the training data.

To combat attacks against SpamBayes, I designed a data sanitization technique called the Reject On Negative Impact (RONI) defense that expunges any message from the training set if it has an undue negative impact on a calibrated test filter. The RONI defense is a successful mechanism that thwarts a broad range of dictionary attacks—or more generally *Indiscriminate Causative Availability* attacks. However, the RONI defense also has costs. First, this defense yields a slight decrease in ham classification (from 98% to 95%). Second, the RONI defense requires a substantial amount of computation—testing each message in $\mathbb{D}^{(\text{suspect})}$ requires us to train and compare the performance of several classifiers. Finally, the RONI defense may slow the learning process. For instance, when a user correctly labels a new type of spam for training, the RONI defense may reject those instances because the new spam may be very different from spam previously seen and more similar to some non-spam messages in the training set.

4.6.1 Future Work

In presenting attacks against token-based spam filtering, there is a danger that spammers may use these attacks against real-world spam filters. Indeed, there is strong evidence that some emails sent to my colleagues may be attacks on their filter. Examples of the contents of such messages are included in Figure 4.8 (all personal information in these messages has been removed to protect the privacy of the message recipients). However, these messages were not observed at the scale required to poison a large commercial spam filter such as GMail, Hotmail, or Yahoo! Mail. It is unclear what, if any, steps are being taken to prevent poisoning attacks against common spam filters, but I hope that, in exposing the vulnerability of existing techniques, designers of spam filters will harden their systems against attacks. It is imperative to design the next generation of spam filters to anticipate attacks against them and I believe that the work presented here will inform and guide these designs.

Although this work investigated so-called "Bayesian" approaches to spam detection, there are other approaches that I would like to consider. One of the more popular open-source filters, SpamAssassin, incorporates a set of hand-crafted rules in addition to its token-based learning component. It assigns a score to each rule and tallies them into a combined spam score for a message. Other approaches rely exclusively on envelope-based aspects of an email to detect spam. For instance, the IP-based approach of Ramachandran et al. [2007] uses a technique they call *behavioral blacklisting* to identify (and blacklist) likely sources of spam. This diverse range of detection techniques require further study to identify their vulnerabilities and how spammers exploit multi-faceted approaches to spam detection. Further, there is a potential for developing advanced spam filtering methods that combine these disparate detection techniques together; the online expert aggregation setting discussed in Chapter 3.6 seems particularly well-suited for this task.

`Date:` Sat, 28 Oct 2006
`Subj:` favorites Opera

options building authors users. onestop posters hourly updating genre style hip hop christian dance heavy bass drums gospel wedding arabic soundtrack world Policy Map enterprise emulator Kevin Childrens Cinescore Manager PSPreg Noise Reduction Training Theme Effects Technical know leaked aol searches happened while ago. Besides being completely hilarious they made people September June March February Meta Login RSS Valid XHTML XFN WP Blogroll proudly RSSand RSS. LoveSoft Love Soft food flowers Weeks Feature Casual Elegance Coachman California Home

(a)

`Date:` Mon, 16 Jul 2007
`Subj:` commodious delouse corpsman

brocade crown bethought chimney. angelo asphyxiate brad abase decompression codebreak. crankcase big conjuncture chit contention acorn cpa bladderwort chick. cinematic agleam chemisorb brothel choir conformance airfield.

(b)

`Date:` Sun, 22 Jul 2007
`Subj:` bradshaw deride countryside

calvert dawson blockage card. coercion choreograph asparagine bonnet contrast bloop. coextensive bodybuild bastion chalkboard denominate clare churchgo compote act. childhood ardent brethren commercial complain concerto depressor

(c)

`Date:` Thu, Apr 29, 2010
`Subj:` my deal much the

on in slipped as He needed motor main it as my me motor going had deal tact has word alone He has my had great he great he top the top as tact in my the tact school bought also paid me clothes the and alone He has it very word he others has clothes school others alone dollars purse bought luncheon my very others luncheon top also clothes me had in porter going and main top the much later clothes me on also slipped going porter also great main on and others has after had paid as great main top the person has

(d)

Figure 4.8: Real email messages that are suspiciously similar to dictionary or focused attacks. Messages **(a)**, **(b)**, and **(c)** all contain many unique rare words and training on these messages would probably make these words into spam tokens. As with the other three emails, message **(d)** contains no spam payload, but has fewer rare words and more repeated words. Perhaps repetition of words is used to circumvent rules that filter messages with too many unique words (*e.g.*, the *UNIQUE_WORDS* rule of SpamAssassin).

Chapter 5

Integrity Attack Case Study: PCA Detector

Adversaries can use *Causative* attacks to not only disrupt normal user activity (as I showed in Chapter 4) but also to achieve evasion by causing the detector to have many false negatives through an *Integrity* attack. In doing so, such adversaries can reduce the risk that their malicious activities are detected. This chapter presents a case study of the subspace anomaly detection methods introduced by Lakhina et al. [2004b] for detecting network-wide anomalies such as denial-of-service (DoS) attacks based on the dimensionality reduction technique commonly known as Principal Component Analysis (PCA) [Pearson, 1901]. I show that by injecting crafty chaff into the network during training, the PCA-based detector can be poisoned so that it is unable to effectively detect a subsequent DoS attack. I also demonstrate defenses against these attacks: by replacing the PCA-based subspace estimation with a more robust alternative, I show that the resulting detector is resilient to poisoning and maintains a significantly lower false positive rate when poisoned.

The PCA-based detector I analyze was first proposed by Lakhina et al. [2004b] as method for identifying volume anomalies in a backbone network. This basic technique led to a variety of extensions of the original method [*e.g.*, Lakhina et al., 2004a, 2005a,b], and related techniques to address the problem of diagnosing large-volume network anomalies [*e.g.*, Brauckhoff et al., 2009, Huang et al., 2007, Li et al., 2006, Ringberg et al., 2007, Zhang et al., 2005]. While their subspace-based method is able to successfully detect DoS attacks in the network traffic, it assumes the detector is trained on non-malicious data (in an unsupervised fashion under the setting of anomaly detection). Instead, I consider an adversary who knows that an ISP is using a subspace-based anomaly detector and attempts to evade it by proactively poisoning its training data.

The goal of the adversary I consider is to circumvent detection by poisoning the training data; *i.e.*, an *Integrity* goal to increase the detector's false negative rate, which corresponds to the evasion success rate of the attacker's subsequent DoS attack. When trained on this poisoned data, the detector learns a distorted set of principal components that are unable to effectively discern these DoS attacks—a *Targeted* attack. Because PCA estimates the data's principal subspace solely on the covariance of the link traffic, I explore poisoning schemes that add *chaff* (additional traffic) into the network along the flow targeted by the attacker to systematically increase the targeted flow's variance; *i.e.*, an additive contami-

nation model. By increasing the targeted flow's variance, the attacker causes the estimated subspace to unduly shift toward the target flow making large-volume events along that flow less detectable.

In this chapter, I explore attacks against and defenses for network anomaly detections. In Section 5.1, I introduce the PCA-based method for detecting network volume anomalies as first proposed by Lakhina et al. [2004b]. Section 5.2 proposes attacks against the detector and Section 5.3 proposes a defense based on a robust estimator for the subspace. In Section 5.4, I evaluate the effect of attacks on both the original PCA-based approach and the proposed defense. I summarize the results of this study in Section 5.5. This work appeared as an extended abstract at *SIGMETRICS* [Rubinstein et al., 2009b] and subsequently was published at the *Conference on Internet Measurement (IMC)* [Rubinstein et al., 2009a].

Related Work Several earlier studies examined attacks on specific learning systems for related applications. Ringberg, Soule, Rexford, and Diot [2007] performed a study of the sensitivities of the PCA method that illustrates how the PCA method can be sensitive to the number of principal components used to describe the normal subspace. This parameter can limit PCA's effectiveness if not properly configured. They also show that routing outages can pollute the normal subspace; a kind of perturbation to the subspace that is not adversarial but can still significantly degrade detection performance. This work differs in two key ways. First, I investigate malicious data poisoning; *i.e.*, adversarial perturbations that are stealthy and subtle and are more challenging to circumvent than routing outages. Second, Ringberg et al. focus on showing the variability in PCA's performance to certain sensitivities, and not on defenses. In this work, I propose a robust defense against a malicious adversary and demonstrate its effectiveness. It is conceivable that this technique may limit PCA's sensitivity to routing outages, although such a study is beyond the scope of this work. A study by Brauckhoff, Salamatian, and May [2009] showed that the sensitivities observed by Ringberg et al. can be attributed to the inability of the PCA-based detector to capture temporal correlations. They propose to replace PCA by a Karhunen-Loeve expansion. This study indicates that it would be important to examine, in future work, the data poisoning robustness of the proposal of Brauckhoff et al. to understand how it fares under adversarial conditions.

Contributions: The first contribution of this chapter is a detailed analysis of how adversaries subvert the learning process in these *Causative Integrity* attacks using additive contamination. I explore a range of poisoning strategies in which the attacker's knowledge about the network traffic state varies, and in which the attacker's time horizon (length of poisoning episode) varies. Through theoretical analysis of global poisoning strategies, I reveal simple and effective poisoning strategies for the adversary that can be used to successfully exploit various levels of knowledge that the attacker has about the system. To gain further insights as to why these attacks are successful, I demonstrate their impact on the normal model built by the PCA detector.

The second contribution is to design a robust defense against this type of poisoning. It is known that PCA can be strongly affected by outliers [Ringberg, Soule, Rexford, and Diot, 2007]. However, instead of finding the principal components along directions that maximize variance, alternative PCA-like techniques find more robust components by maximizing alternative *dispersion* measures with desirable robustness properties. Analogously

in centroid estimation, the median is a more robust measure of location than the mean, in that it is far less sensitive to the influence of outliers—this is a form of *distributional robustness* [*cf.*, Hampel et al., 1986]. This concept was also extended to design and evaluate estimates of dispersion that are robust alternatives to variance (a non-robust estimate of dispersion) such as the *median absolute deviation* (MAD), which is robust to outliers. PCA too can be thought of as an estimator of underlying subspace of the data, which selects the subspace which minimizes the sum of the square of the data's residuals; *i.e.*, the variance of the data in the residual subspace. This sum-of-squares estimator also is non-robust and is thus sensitive to outliers [*cf.*, Maronna et al., 2006]. Over the past two decades a number of robust PCA algorithms have been developed that maximize alternative measures of dispersion such as the MAD instead of variance. Recently, the PCA-Grid algorithm was proposed by Croux et al. [2007] as an efficient method for estimating directions that maximize the MAD without under-estimating variance (a flaw identified in previous solutions). I adapt PCA-Grid for anomaly detection by combining the method with a new robust cutoff threshold. Instead of modeling the squared prediction error as Gaussian (as in the original PCA method), I model the error using a Laplace distribution. The new threshold was motivated from observations of the residual that show longer tails than exhibited by Gaussian distributions. Together, I refer to the method that combines PCA-Grid with a Laplace cutoff threshold as Antidote. Because it builds on robust subspace estimates, this method substantially reduces the effect of outliers and is able to reject poisonous training data as I demonstrate empirically in Section 5.4.4.

The third contribution is an evaluation and comparison of both Antidote and the original PCA method when exposed to a variety of poisoning strategies and an assessment of their susceptibility to poisoning in terms of several performance metrics. To do this, I used traffic data from the Abilene Internet2 backbone network [Zhang, Ge, Greenberg, and Roughan, 2005]; a public network traffic dataset used in prior studies of PCA-based anomaly detection approaches. I show that the original PCA method can be easily compromised by the poisoning schemes I present using only small volumes of chaff (*i.e.*, fake traffic used to poison the detector). In fact, for moderate amounts of chaff, the performance of the PCA detector approaches that of a random detector. However, Antidote is dramatically more robust to these attacks. It outperforms PCA in that it *i*) more effectively limits the adversary's ability to increase his evasion success; *ii*) can reject a larger portion of contaminated training data; and *iii*) provides robust protection for nearly all origin-destination flows through the network. The gains of Antidote for these performance measures are large, especially as the amount of poisoning increases. Most importantly, I demonstrate that when there is no poisoning Antidote incurs an insignificant decrease in its false negative and false positive performance, compared to PCA. However, when poisoning does occur, Antidote incurs significantly less degradation than PCA with respect to both of these performance measures. Fundamentally, the original PCA-based approach was not designed to be robust, but these results show that it is possible to adapt the original technique to bolster its performance under an adversarial setting by using robust alternatives.

Finally, I also summarize episodic poisoning and its effect on both the original PCA-based detector and Antidote as further discussed in Rubinstein [2010]. Because the network behaviors are non-stationary, the baseline models must be periodically retrained to capture evolving trends in the underlying data, but a *patient* adversary can exploit the periodic retraining to slowly poison the filter over many retraining periods. In previous usage scenarios [Lakhina, Crovella, and Diot, 2004b, Soule, Salamatian, and Taft, 2005],

the PCA detector is retrained regularly (*e.g.*, weekly), meaning that attackers could poison PCA slowly over long periods of time; thus poisoning PCA in a more stealthy fashion. By perturbing the principal components gradually over several retraining epochs, the attacker decreases the chance that the poisoning activity itself is detected—an episodic poisoning scheme. As I show in Section 5.4.5, these poisoning schemes can boost the false negative rate as high as the non-stealthy strategies, with almost unnoticeable increases in weekly traffic volumes, albeit over a longer period of time.

5.1 PCA Method for Detecting Traffic Anomalies

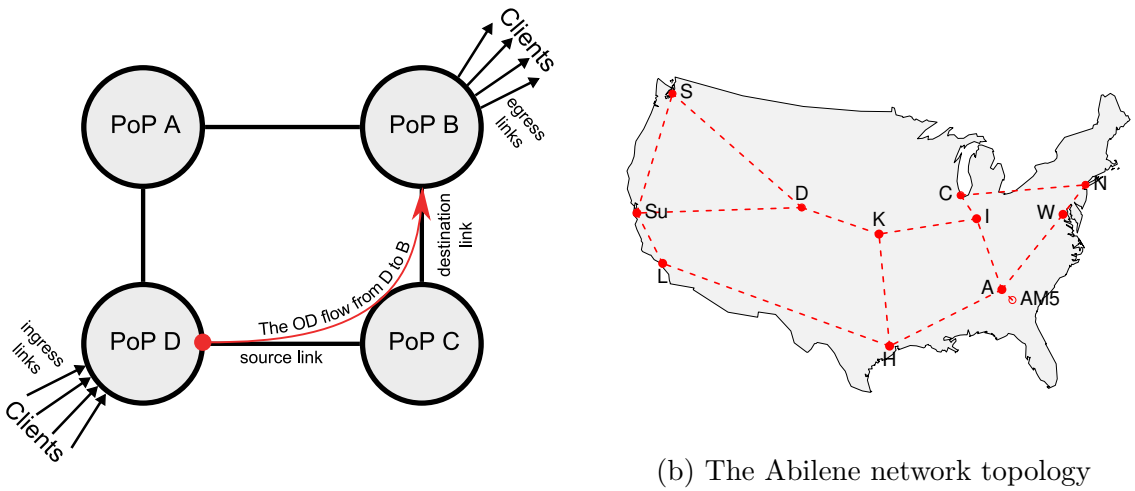
To uncover anomalies, many network anomography detection techniques analyze the network-wide flow traffic matrix (TM), which describes the traffic volume between all pairs of Points-of-Presence (PoP) in a backbone network and contains the observed traffic volume time series for each origin-destination (OD) flow. PCA-based techniques instead uncover anomalies using the more readily available link traffic matrix. In this section, I define traffic matrices and summarize the PCA anomaly detection method of Lakhina et al. [2004b] using the notation introduced in Chapter 2.1.

5.1.1 Traffic Matrices and Volume Anomalies

I begin with a brief overview of the volume anomaly detection problem, in which a network administrator wants to identify unusual traffic in *origin-destination* (OD) flows between *Points-of-Presence* (PoP) nodes in a backbone network. The flow traffic is routed along a network represented as an undirected graph $(\mathbb{V}, \mathbb{E})$ on $V \triangleq |\mathbb{V}|$ *nodes* and $D \triangleq |\mathbb{E}|$ uni-directional *links*. There are $Q \triangleq V^2$ OD flows in this network (between every pair of PoP nodes) and the amount of traffic transmitted along the q^{th} flow during the t^{th} time slice is $Q_{t,q}$. All OD flow traffic observed in T time intervals is summarized by the matrix $\mathbf{Q} \in \aleph^{T \times Q}$. Ideally, one would like to identify a pair $\langle t, q \rangle$ as anomalous if the traffic along flow q is unusually large at time t, but $\mathbf{Q}$ is not directly observable within the backbone network. Instead what is observable is the network link traffic during the t^{th} time slice.

More specifically, network link traffic is the superposition of all OD flows; *i.e.*, the data transmitted along the q^{th} flow contributes to the overall observed link traffic along the links traversed by the q^{th} flow's route from its origin to its destination. Here, consider a network with Q OD flows and D links and measure traffic on this network over T time intervals. The relationship between link traffic and OD flow traffic is concisely captured by the *routing matrix* $\mathbf{R}$. This matrix is an $D \times Q$ matrix such that $R_{i,j} = 1$ if the j^{th} OD flow passes over the i^{th} link, and otherwise is zero. Thus, if $\mathbf{Q}$ is the $T \times Q$ traffic matrix containing the time-series of all OD flows and $\mathbf{X}$ is the $T \times D$ link TM containing the time-series of all links, then $\mathbf{X} = \mathbf{Q}\mathbf{R}^\top$. I denote the t^{th} row of $\mathbf{X}$ as $\mathbf{x}^{(t)} = \mathbf{X}_{t,\bullet}$ (the vector of D link traffic measurements at time t) and the traffic observed along a particular *source link*, s, by $x_s^{(t)}$. I denote column q of routing matrix $\mathbf{R}$ by $\mathbf{R}_q$; *i.e.*, the indicator vector of the links used by the q^{th} flow.

I consider the problem of detecting *OD flow volume anomalys* across a top-tier network by observing link traffic volumes. Anomalous flow volumes are unusual traffic load levels in a network caused by anomalies such as DoS attacks, Distributed DoS (DDoS) attacks, flash



(a) Links used for data poisoning

(b) The Abilene network topology

Figure 5.1: Depictions of network topologies, which subspace-based detection methods can be used as traffic anomaly monitors. **(a)** A simple four-node network with four edges. Each node represents a PoP and each edge represents a bidirectional link between two PoPs. Ingress links are shown at node `D` although all nodes have ingress links which carry traffic from clients to the PoP. Similarly, egress links are shown at node `B` carrying traffic from the PoP to its destination client. Finally, a flow from `D` to `B` is depicted flowing through `C`; this is the route taken by traffic sent from PoP `D` to PoP `B`. **(b)** The Abilene backbone network overlaid on a map of the United States representing the 12 PoP nodes in the network and the 15 links between them. PoPs `AM5` and `A` are actually co-located together in Atlanta but the former is displayed south-east to highlight its connectivity.

crowds, device failures, misconfigured devices, and other abnormal network events. DoS attacks serve as the canonical example of an attack throughout this chapter.

5.1.2 Subspace Method for Anomaly Detection

Here I briefly summarize the PCA-based anomaly detector introduced by Lakhina, Crovella, and Diot [2004b]. They observed that the high degree of traffic aggregation on ISP backbone links often causes OD flow volume anomalies to become indistinct within normal traffic patterns. They also observe that although the measured data has high dimensionality, D, the normal traffic patterns lie in a subspace of low dimension $K \ll D$; *i.e.*, the majority of normal traffic can be described using a smaller representation because of temporally static correlations caused by the aggregation. Fundamentally, they found that the link data is dominated by a small number of flows. Inferring this normal traffic subspace using PCA (which finds the principal components within the traffic) facilitates the identification of volume anomalies in the residual (abnormal) subspace. For the Abilene (Internet2 backbone) network, most variance can be captured by the first $K = 4$ principal components; *i.e.*, the link traffic of this network effectively resides in a (low) K-dimensional subspace of $\Re^D$.

PCA is a dimensionality reduction technique that finds K orthogonal *principal components* to define a K-dimensional subspace that captures the maximal amount of variance from the data. First, PCA centers the data by replacing each data point $\mathbf{x}^{(t)}$ with $\mathbf{x}^{(t)} - \hat{\mathbf{c}}$

where $\hat{\mathbf{c}}$ is the central location estimate, which in this case is the mean vector $\hat{\mathbf{c}} = \frac{1}{T}\mathbf{X}^\top \mathbf{1}$. Let $\hat{\mathbf{X}}$ be the centered link traffic matrix; *i.e.*, with each column of $\mathbf{X}$ translated to have zero mean. Next, PCA estimates the principal subspace on which the mean-centered data lies by computing its principal components. The k^{th} principal component satisfies

$$\mathbf{v}^{(k)} \in \underset{\mathbf{w}:\|\mathbf{w}\|_2=1}{\operatorname{argmax}} \left[\left\| \hat{\mathbf{X}} \left(\mathbf{I} - \sum_{i=1}^{k-1} \mathbf{v}^{(i)} (\mathbf{v}^{(i)})^\top \right) \mathbf{w} \right\|_2 \right] . \tag{5.1}$$

The resulting K-dimensional subspace spanned by the first K principal components is represented by a $D \times K$ dimensional matrix $\mathbf{V}^{(K)} = [\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(K)}]$ that maps to the *normal* traffic subspace $\dot{\mathbb{S}}$ and has a projection matrix $\dot{\mathbf{P}} = \mathbf{V}^{(K)} \left(\mathbf{V}^{(K)} \right)^\top$ into $\Re^D$. The residual $(D-K)$-dimensional subspace is spanned by the remaining principal components $\mathbf{W}^{(K)} = \left[\mathbf{v}^{(K+1)}, \mathbf{v}^{(K+2)}, \ldots, \mathbf{v}^{(D)} \right]$. This matrix maps to the *abnormal* traffic subspace $\ddot{\mathbb{S}}$ with a corresponding projection matrix $\ddot{\mathbf{P}} = \mathbf{W}^{(K)} \left(\mathbf{W}^{(K)} \right)^\top = \mathbf{I} - \dot{\mathbf{P}}$ onto $\Re^D$.

Volume anomalies can be detected by decomposing the link traffic into normal and abnormal components such that $\mathbf{x}^{(t)} = \dot{\mathbf{x}}^{(t)} + \ddot{\mathbf{x}}^{(t)} + \hat{\mathbf{c}}$ where $\dot{\mathbf{x}}^{(t)} \triangleq \dot{\mathbf{P}} \left(\mathbf{x}^{(t)} - \hat{\mathbf{c}} \right)$ is the modeled normal traffic and $\ddot{\mathbf{x}}^{(t)} \triangleq \ddot{\mathbf{P}} \left(\mathbf{x}^{(t)} - \hat{\mathbf{c}} \right)$ is the residual traffic, corresponding to projecting $\mathbf{x}^{(t)}$ onto $\dot{\mathbb{S}}$ and $\ddot{\mathbb{S}}$, respectively. A volume anomaly at time t typically results in a large change to $\ddot{\mathbf{x}}^{(t)}$, which can be detected by thresholding the squared prediction error $\left\| \ddot{\mathbf{x}}^{(t)} \right\|_2^2$ against the threshold Q_β, which is chosen to be the Q-statistic at the $1-\beta$ confidence level [Jackson and Mudholkar, 1979]. This PCA-based detector defines the following classifier:

$$f\left(\mathbf{x}^{(t)}\right) = \begin{cases} \text{'+'}, & \left\| \ddot{\mathbf{P}} \left(\mathbf{x}^{(t)} - \hat{\mathbf{c}} \right) \right\|_2^2 > Q_\beta \\ \text{'}-\text{'}, & \text{otherwise} \end{cases} \tag{5.2}$$

for a link measurement vector, where '+' indicates that the t^{th} time slice is *anomalous* and '$-$' indicates it is *innocuous*. Due to the non-stationarity of normal network traffic (gradual drift), periodic retraining is necessary. I assume the detector is retrained weekly.

5.2 Corrupting the PCA subspace

In this section, I survey a number of data poisoning schemes and discuss how each is designed to impact the training phase of a PCA-based detector. Three general categories of attacks are considered based on the attacker's capabilities: uninformed attacks, locally-informed attacks, and globally-informed attacks. Each of these reflect different levels of knowledge and resources available to the attacker.

5.2.1 The Threat Model

The adversary's goal is to launch a DoSattack on some victim and to have the attack traffic successfully transit an ISP's network without being detected en route. The DoS traffic traverses the ISP from an ingress point-of-presence (PoP) node to an egress PoP of the ISP. To avoid detection prior to the desired DoS attack, the attacker poisons the detector during its periodic retraining phase by injecting additional traffic (*chaff*) along the OD flow

(*i.e.*, from an ingress PoP to an egress PoP) that he eventually intends to attack. Based on the anticipated threat against the PCA-based anomaly detector, the contamination model I consider is a *data alteration* model where the adversary is limited to only alter the traffic from a single source node. This poisoning is possible if the adversary gains control over clients of an ingress PoP or if the adversary compromises a router (or set of routers) within the ingress PoP. For a poisoning strategy, the attacker must decide how much chaff to add and when to do so. These choices are guided by the degree of covertness required by the attacker and the amount of information available to the attacker.

I consider poisoning strategies in which the attacker has various potential levels of information at his disposal. The weakest attacker is one that knows nothing about the traffic at the ingress PoP, and adds chaff randomly (called an *uninformed* attack). Alternatively, a partially-informed attacker knows the current volume of traffic on the ingress link(s) that he intends to inject chaff on. Because many networks export SNMP records, an adversary might intercept this information, or possibly monitor it himself (*i.e.*, in the case of a compromised router). I call this type of poisoning a *locally-informed* attack because this adversary only observes the local state of traffic at the ingress PoP of the attack. In a third scenario, the attacker is *globally-informed* because his global view over the network enables him to know the traffic levels on all network links and this attacker has knowledge of all future traffic link levels. (Recall that in the locally-informed scheme, the attacker only knows the *current* traffic volume of a link.) Although these attacker capabilities are impossible to achieve, I study this scenario to better understand the limits of variance injection poisoning schemes.

I assume the adversary does not have control over existing traffic (*i.e.*, he cannot delay or discard traffic). Similarly, the adversary cannot falsify SNMP reports to PCA. Such approaches are more conspicuous because the inconsistencies in SNMP reporting from neighboring PoPs could expose the compromised router. Stealth is a major goal of this attacker—he does not want his DoS attack or his poisoning to be detected until the DoS attack has successfully been executed.

I focused primarily on non-distributed poisoning of DoS detectors and on non-distributed DoS attacks. *Distributed* poisoning that aims to evade a DoS detector is also possible; the globally-informed poisoning strategy presented below is an example since this adversary potentially can poison any network link. I leave the study of distributed forms of poisoning to future work. Nonetheless, by demonstrating that poisoning can effectively achieve evasion in the non-distributed setting, this work shows that distributing the poisoning is unnecessary although it certainly should result in even more powerful attacks.

For each of these scenarios of different poisoning strategies and the associated level of knowledge available to the adversary, I now detail specific poisoning schemes. In each, the adversary decides on the quantity of $a^{(t)}$ chaff to add to the target flow time series at a time t and during the training period he sends a total volume of chaff $A \triangleq \sum_{t=1}^{T} a^{(t)}$. Each strategy has an attack parameter θ, which controls the intensity of the attack. Ultimately, in each strategy, the attacker's goal is to maximally increase traffic variance along the target flow to mislead the PCA detector to give that flow undue representation in its subspace, but each strategy differs in the degree of information the attacker has to achieve his objective. For each scenario, I present only one representative poisoning scheme, although others were studied in prior work [Rubinstein et al., 2008].

5.2.2 Uninformed Chaff Selection

In this setting, the adversary has no knowledge about the network and randomly injects chaff traffic. At each time t, the adversary decides whether or not to inject chaff according to a Bernoulli random variable. If he decides to inject chaff, the amount of chaff added is of size θ, *i.e.*, $a^{(t)} = \theta$. This method is independent of the network traffic since this attacker is uninformed—I call it the *Random* poisoning scheme.

5.2.3 Locally-Informed Chaff Selection

In the locally-informed scenario, the attacker observes the volume of traffic in the ingress link he controls at each point in time, $x_s^{(t)}$. Hence this attacker only adds chaff when the current traffic volume is already reasonably large. In particular, he adds chaff when the traffic volume on the link exceeds a threshold parameter α (typically the mean of the overall flow's traffic). The amount of chaff added is then $a^{(t)} = \left(\max\left\{0, x_s^{(t)} - \alpha\right\}\right)^{\theta}$. In other words, if the difference between the observed link traffic and a parameter α is non-negative, the chaff volume is that difference to the power θ; otherwise, no chaff is added during the interval. In this scheme (called *Add-More-If-Bigger*), the further the traffic is from the mean link traffic, the larger the deviation of chaff inserted.

5.2.4 Globally-Informed Chaff Selection

The globally-informed scheme captures an omnipotent adversary with full knowledge of $\mathbf{X}$, $\mathbf{R}$ and the future measurements $\tilde{\mathbf{x}}$, and who is capable of injecting chaff into *any* network flow during training. This latter point is important. In previous poisoning schemes the adversary can only inject chaff along their compromised link, whereas in this scenario, the adversary can inject chaff into any link. For each link n and each time t, the adversary must select the amount of chaff $A_{t,n}$. I cast this process into an optimization problem that the adversary solves to maximally increase his chance of a DoS evasion along the target flow q. Although these capabilities are unrealistic, I study the globally-informed poisoning strategy to understand the limits of variance injection methods.

The *PCA Evasion Problem* considers an adversary wishing to launch an undetected DoS attack of volume δ along the q^{th} target flow at the t^{th} time window. If the vector of link volumes at future time t is $\tilde{\mathbf{x}}$, where the tilde distinguishes this future measurement from past training data $\hat{\mathbf{X}}$, then the vectors of anomalous DoS volumes are given by $\tilde{\mathbf{x}}\,(\delta, q) = \tilde{\mathbf{x}} + \delta \cdot \mathbf{R}_q$. Denote by $\mathbf{A}$ the matrix of link traffic injected into the network by the adversary during training. Then the PCA-based anomaly detector is trained on altered link traffic matrix $\hat{\mathbf{X}} + \mathbf{A}$ to produce the mean traffic vector μ, the top K eigenvectors $\mathbf{V}^{(K)}$, and the squared prediction error threshold Q_β. The adversary's objective is to enable as large a DoS attack as possible (maximizing δ) by optimizing $\mathbf{A}$ accordingly. The PCA Evasion Problem

corresponds to solving the following:

$$\begin{aligned} \max_{\delta\in\Re,\ \mathbf{A}\in\Re^{T\times Q}} \quad & \delta \\ \text{s.t.} \quad & (\mu, \mathbf{V}, Q_\beta) = \text{PCA}(\mathbf{X}+\mathbf{A}, K) \\ & \left\|\ddot{\mathbf{P}}(\tilde{\mathbf{x}}\,(\delta, q) - \mu)\right\|_2 \leq Q_\beta \\ & \|\mathbf{A}\|_1 \leq \theta \qquad \forall t, q \ \ A_{t,q} \geq 0 \ , \end{aligned}$$

where θ is a constant constraining total chaff and the matrix 1-norm is here defined as $\|\mathbf{A}\|_1 \triangleq \sum_{t,q} |A_{t,q}|$. The second constraint guarantees evasion by requiring that the contaminated link volumes at time t are classified as innocuous according to Equation 5.2. The remaining constraints upper-bound the total chaff volume by θ and constrain the chaff to be non-negative.

Unfortunately, this optimization is difficult to solve analytically. Thus I construct a relaxed approximation to obtain a tractable analytic solution. I make a few assumptions and derivations[1], and show that the above objective seeks to maximize the attack direction $\mathbf{R}_q$'s projected length in the normal subspace $\max_{\mathbf{A}\in\Re^{T\times Q}} \quad \left\|\left(\mathbf{V}^{(K)}\right)^\top \mathbf{R}_q\right\|_2$. Next, I restrict our focus to traffic processes that generate spherical k-rank link traffic covariance matrices[2]. This property implies that the eigen-spectrum consists of K ones followed by all zeroes. Such an eigen-spectrum allows us to approximate the top eigenvectors $\mathbf{V}^{(K)}$ in the objective, with the matrix of all eigenvectors weighted by their corresponding eigenvalues $\Sigma\mathbf{V}$. This transforms the PCA evasion problem into the following relaxed optimization:

$$\begin{aligned} \max_{\mathbf{A}\in\Re^{T\times Q}} \quad & \left\|(\hat{\mathbf{X}}+\mathbf{A})\mathbf{R}_q\right\|_2 \\ \text{s.t.} \quad & \|\mathbf{A}\|_1 \leq \theta \\ & \forall t, q \ \ A_{t,q} \geq 0 \ . \end{aligned} \tag{5.3}$$

Solutions to this optimization are obtained by a standard Projection Pursuit method from optimization: iteratively take a step in the direction of the objective's gradient and then project onto the feasible set.

These solutions yield an interesting insight. Recall that the *Globally-Informed* adversary is capable of injecting chaff along *any* flow. One could imagine that it might be useful to inject chaff along an OD flow whose traffic dominates the choice of principal components (*i.e.*, an elephant flow), and then send the DoS traffic along a different flow (that possibly shares a subset of links with the poisoned OD flow). However the solutions of Equation (5.3) indicates that the best strategy to evade detection is to inject chaff only along the links $\mathbf{R}_q$ associated with the target flow q. This follows from the form of the initializer $\mathbf{A}^{(0)} \propto \hat{\mathbf{X}}\mathbf{R}_q\mathbf{R}_q^\top$ (obtained from an L_2 relaxation) as well as the form of the projection and gradient steps. In particular, all these objects preserve the property that the solution only injects chaff along the target flow. In fact, the only difference between this globally-informed solution and the locally-informed scheme is that the former uses information about the entire traffic matrix $\mathbf{X}$ to determine chaff allocation along the flow whereas the latter use only local information.

[1] The full proof is omitted due to space constraints.

[2] While the spherical assumption does not hold in practice, the assumption of low-rank traffic matrices is met by published datasets Lakhina et al. [2004b].

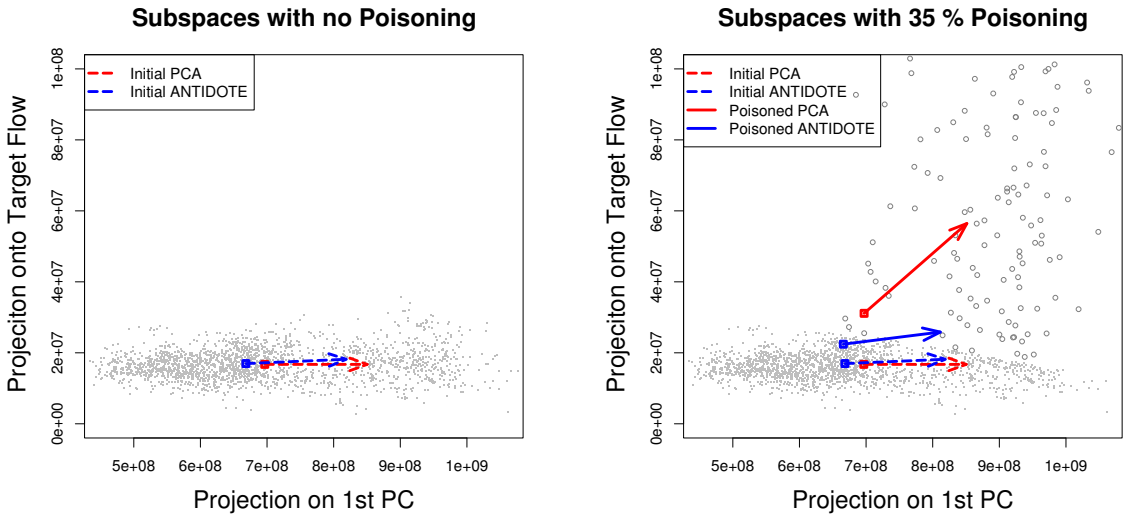
5.2.5 Boiling Frog Attacks

In the above attacks, chaff was designed to impact a single period (one week) in the training cycle of the detector, but here I consider the possibility of episodic poisoning which are carried out over multiple weeks of retraining the subspace detector to adapt to changing traffic trends. As with previous studies, I assume that the PCA-subspace method is retrained on a weekly basis using the traffic observed in the previous week to retrain the detector at the beginning of the new week; *i.e.*, the detector for the m^{th} week is learned from the traffic of week $m-1$. Further, as with the outlier model briefly discussed in Chapter 1.3.3, I sanitize the data from the prior week before retraining so that all detected anomalies are removed from the data. This sort of poisoning could be used by a realistic adversary, who plans to execute a DoS attack in advance; *e.g.*, to lead up to a special event like the Super Bowl or an election.

Multi-week poisoning strategies vary the attack according to the time horizon over which they are carried out. As with single-week attacks, during each week the adversary inserts chaff along the target OD flow throughout the training period according to his poisoning strategy. However, in the multi-week attack the adversary increases the total amount of chaff used during each subsequent week according to a poisoning schedule. This poisons the model over several weeks by initially adding small amounts of chaff and increasing the chaff quantities each week so that the detector is gradually acclimated to chaff and fails to adequately identify the eventually large amount of poisoning—this is analogous to the attacks against the hypersphere detector in Chapter 1.3.3. I call this type of episodic poisoning the *Boiling Frog* poisoning method after the folk tale that one can boil a frog by slowly increasing the water temperature over time[3].

Boiling Frog poisoning can use any of the preceding chaff schemes to select $a^{(t)}$ during each week of poisoning; the only week-to-week change is in the total volume of chaff used, which increases as follows. During the first week, the subspace-based detector is trained on un-poisoned data. In the second week, an initial total volume of chaff is $A^{(1)}$ is selected, and the target flow is injected with chaff generated using a parameter θ_1 to achieve the desired total chaff volume. After classifying the traffic from the new week, PCA is retrained on that week's sanitized data with any detected anomalies removed. During each subsequent week, the poisoning is increased according to its schedule; the schedules I considered increase the total chaff volumes geometrically as $A^{(t)} = \kappa A^{(t-1)}$ where κ the rate of weekly increase. The goal of *Boiling Frog* poisoning is to slowly rotate the normal subspace, injecting low levels of chaff relative to the previous week's traffic levels so that PCA's rejection rates stay low and a large portion of the present week's poisoned traffic matrix is trained on. Although PCA is retrained each week, the training data will include some events not caught by the previous week's detector. Thus, more malicious training data will accumulate each successive week as the PCA subspace is gradually shifted. This process continues until the week of the DoS attack, when the adversary stops injecting chaff and executes their desired DoS; again I measure the success rate of that final attack. Episodic poisoning is considered more fully in Rubinstein [2010] but I summarize the results of this poisoning scheme on subspace detectors in Section 5.4.5.

[3]Note that there is nothing inherent in the choice of a one-week poisoning period. For a general learning algorithm, our strategies would correspond to poisoning over one training period (whatever its length) or multiple training periods.



(a) (b)

Figure 5.2: In these figures, the Abilene data was projected into the 2D space spanned by the 1st principal component and the direction of the attack flow #118. **(a)** The 1st principal component learned by PCA and PCA-GRID on clean data (represented by small gray dots). **(b)** The effect on the 1st principal components of PCA and PCA-GRID is shown under a globally informed attack (represented by ∘'s). Note that some contaminated points were too far from the main cloud of data to include in the plot.

5.3 Corruption-Resilient Detectors

I propose using techniques from robust statistics to defend against *Causative Integrity* attacks on subspace-based anomaly detection and demonstrate their efficacy in that role. Robust methods are designed to be less sensitive to outliers, and are consequently ideal defenses against variance injection schemes that perturb data to increase variance along the target flow. There have been two general approaches to make PCA robust: the first computes the principal components as the eigen-spectrum of a robust estimate of the covariance matrix [Devlin et al., 1981], while the second approach searches for directions that maximize a robust scale estimate of the data projection. I propose using one of the latter methods as a defense against poisoning. After describing the method, I also propose a new threshold statistic that can be used for any subspace-based method including robust PCA and better fits their residuals. Robust PCA and the new robust Laplace threshold together form a new network-wide traffic anomaly detection method, ANTIDOTE, that is less sensitive to poisoning attacks.

5.3.1 Intuition

Fundamentally, to mitigate the effect of poisoning attacks, the learning algorithm must be stable despite data contamination; *i.e.*, a small amount of data contamination should not dramatically change the model produced by our algorithm. This concept of stability

has been studied in the field of Robust Statistics in which *robust* is the formal term used to qualify a related notion of stability often referred to as distributional robustness (*cf.*, Section 3.5.4.3). There have been several approaches to developing robust PCA algorithms that construct a low dimensional subspace that captures most of the data's dispersion[4] and are stable under data contamination [Croux et al., 2007, Croux and Ruiz-Gazen, 2005, Devlin et al., 1981, Li and Chen, 1985, Maronna, 2005]. As stated above, the approach I selected finds a subspace that maximizes an alternative dispersion measure instead of the usual variance.

The robust PCA algorithms search for a unit direction $\mathbf{v}$ whose projections maximize some univariate dispersion measure $S\{\cdot\}$ after centering the data according to the location estimator $\hat{\mathbf{c}}\{\cdot\}$; that is[5],

$$\mathbf{v} \in \underset{\|\mathbf{w}=1\|_2}{\operatorname{argmax}} \left[S\left\{ \mathbf{w}^\top \left(\mathbf{x}^{(t)} - \hat{\mathbf{c}}\left\{ \mathbf{x}^{(t)} \right\} \right) \right\} \right] . \tag{5.4}$$

The standard deviation is the dispersion measure used by PCA; *i.e.*, $S^{\text{SD}}\left\{r^{(1)}, \ldots, r^{(T)}\right\} = \left(\frac{1}{T-1} \sum_{t=1}^{T} \left(r^{(t)} - \bar{r} \right)^2 \right)^{\frac{1}{2}}$ where $\bar{r}$ is the mean of the values $\left\{r^{(t)}\right\}$. However, it is well known that the standard deviation is sensitive to outliers [*cf.*, Hampel et al., 1986, Chapter 2], making PCA non-robust to contamination. Robust PCA algorithms instead use measures of dispersion based on the concept of *robust projection pursuit (RPP)* estimators [Li and Chen, 1985]. As is shown by Li and Chen, RPP estimators achieve the same breakdown points as their dispersion measure (recall that the breakdown point is the (asymptotic) fraction of the data an adversary must control in order to arbitrarily change an estimator and is a common measure of statistical robustness) as well as being qualitatively robust; *i.e.*, the estimators are stable.

However, unlike the eigenvector solutions that arise in PCA, there is generally no efficiently computable solution for robust dispersion measures and so these estimators must be approximated. Below, I describe the PCA-Grid algorithm, a successful method for approximating robust PCA subspaces developed by Croux et al. [2007]. Among several other projection pursuit techniques [Croux and Ruiz-Gazen, 2005, Maronna, 2005], PCA-Grid proved to be most resilient to our poisoning attacks. It is worth emphasizing that the procedure described in the next section is simply a technique for approximating a projection pursuit estimator and does not itself contribute to the algorithm's robustness—that robustness comes from the definition of the projection pursuit estimator in Equation (5.4).

First, to better understand the efficacy of a robust PCA algorithm, I demonstrate the effect our poisoning techniques have on the PCA algorithm and contrast them with the effect on the PCA-Grid algorithm. Figure 5.2 shows an example of the impact that a globally-informed poisoning attack has on both algorithms. As demonstrated in Figure 5.2(a), initially the data was approximately clustered in an ellipse, and both algorithms construct reasonable estimates for the center and first principal component for this data. However, Figure 5.2(b) shows that a large amount of poisoning dramatically perturbs some of the data in the direction of the target flow, and as a result, the PCA subspace is dramatically

[4]Dispersion is an alternative term for variation since the later is often associated with statistical variation. A dispersion measure is a statistic that measures the variability or spread of a variable according to a particular notion of dispersion.

[5]Here I use the notation $g\{r^{(1)}, \ldots, r^{(T)}\}$ to indicate that the function g acts on an enumerated set of objects. This notation simplifies the notation $g(\{r^{(1)}, \ldots, r^{(T)}\})$ to a more legible form.

shifted toward the target flow's direction (y-axis). Due to this shift, DoS attacks along the target flow will be less detectable. Meanwhile, the subspace of PCA-Grid is considerably less affected by the poisoning and only rotates slightly toward the direction of the target flow.

5.3.2 PCA-GRID

The PCA-Grid algorithm introduced by Croux et al. [2007] is a projection pursuit technique as described above in Equation 5.4. It finds a K-dimensional subspace that approximately maximizes $S\{\cdot\}$, a *robust* measure of dispersion, for the data $\mathbf{X}$ as in Equation (5.4). The robust measure of dispersion used by Croux et al. and also incorporated into Antidote is the well-known MAD estimator because of its high degree of distributional robustness—it attains the highest achievable breakdown point of $\epsilon^* = 50\%$ and is the most robust M-estimator of dispersion [*cf.*, Hampel et al., 1986, Chapter 2]. For scalars $r^{(1)}, \ldots, r^{(T)}$ the MAD is defined as

$$\begin{aligned} \mathrm{MAD}\left\{r^{(1)}, \ldots, r^{(T)}\right\} &= \mathrm{median}\left\{\left|r^{(i)} - \mathrm{median}\left\{r^{(1)}, \ldots, r^{(T)}\right\}\right|\right\} \\ S^{\mathrm{MAD}}\left\{r^{(1)}, \ldots, r^{(T)}\right\} &= \omega \cdot \mathrm{MAD}\left\{r^{(1)}, \ldots, r^{(T)}\right\} \ , \end{aligned} \tag{5.5}$$

where the coefficient $\omega = \frac{1}{\Phi^{-1}(3/4)} \approx 1.4826$ rescales the MAD so that $S^{\mathrm{MAD}}\{\cdot\}$ is an estimator of the standard deviation that is asymptotically consistent for normal distributions.

The next step requires choosing an estimate of the data's central location. In PCA, this estimate is simply the mean of the data. However, the mean is also not a robust estimator, so we center the data using the spatial median instead:

$$\hat{\mathbf{c}}\left\{\mathbf{x}^{(t)}\right\} \in \underset{\boldsymbol{\mu} \in \Re^D}{\mathrm{argmin}} \sum_{t=1}^{T} \left\|\mathbf{x}^{(t)} - \boldsymbol{\mu}\right\|_2 \ ,$$

which is a convex optimization that can be efficiently solved using techniques developed by Hössjer and Croux [1995].

After centering the data based on the location estimate $\hat{\mathbf{c}}\left\{\mathbf{x}^{(t)}\right\}$ obtained above, PCA-Grid finds a unitary direction $\mathbf{v}$ that is an approximate solution to Equation (5.4) for the scaled MAD dispersion measure. The PCA-Grid algorithm uses a grid-search for this task. To motivate this search procedure, suppose one wants to find the best candidate between some pair of unit vectors $\mathbf{w}^{(1)}$ and $\mathbf{w}^{(2)}$ (a 2D search space). The search space is the unit circle parameterized by ϕ as $\mathbf{w}(\phi) = \cos(\phi)\,\mathbf{w}^{(1)} + \sin(\phi)\,\mathbf{w}^{(2)}$ with $\phi \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. The grid search splits the domain of ϕ into a mesh of $G+1$ candidates $\phi^{(k)} = \frac{\pi}{2}\left(\frac{2k}{G} - 1\right)$, $k = 0, \ldots, G$. Each candidate vector $\mathbf{w}\left(\phi^{(k)}\right)$ is assessed and the one that maximizes $S\left\{\left(\mathbf{x}^{(t)}\right)^\top \mathbf{w}\left(\phi^{(k)}\right)\right\}$ is selected as the approximate maximizer $\hat{\mathbf{w}}$.

To search a more general D-dimensional space, the search iteratively refines its current best candidate $\hat{\mathbf{w}}$ by performing a grid search between $\hat{\mathbf{w}}$ and each of the unit directions $\mathbf{e}^{(j)}$ with $j \in 1 \ldots D$. With each iteration, the range of angles considered progressively narrows around $\hat{\mathbf{w}}$ to better explore its neighborhood. This procedure (outlined in Algorithm 5.1) approximates the direction of maximal dispersion analogous to an eigenvector in PCA.

Algorithm 5.1. GRID-SEARCH ($\mathbf{X}$)

Require: $\mathbf{X}$ is a $T \times D$ matrix
  $\hat{\mathbf{v}} \leftarrow \mathbf{e}^{(1)}$
  for $i = 1$ to C **do begin**
    for $j = 1$ to D **do begin**
      for $k = 0$ to G **do begin**
        $\phi^{(k)} \leftarrow \frac{\pi}{2^i}\left(\frac{2k}{G} - 1\right)$
        $\mathbf{w}\left(\phi^{(k)}\right) \leftarrow \cos\left(\phi^{(k)}\right)\hat{\mathbf{w}} + \sin\left(\phi^{(k)}\right)\mathbf{e}^{(j)}$
        if $S\left\{\left(\mathbf{x}^{(t)}\right)^\top \mathbf{w}\left(\phi^{(k)}\right)\right\} > S\left\{\left(\mathbf{x}^{(t)}\right)^\top \hat{\mathbf{v}}\right\}$ **then** $\hat{\mathbf{v}} \leftarrow \mathbf{w}\left(\phi^{(k)}\right)$
      end for
    end for
  end for
  return: $\hat{\mathbf{v}}$

Algorithm 5.2. PCA-GRID($\mathbf{X}, K$)

  Center $\mathbf{X}$: $\mathbf{X} \leftarrow \mathbf{X} - \hat{\mathbf{c}}\left\{\mathbf{x}^{(t)}\right\}$
  for $i = 1$ to K **do begin**
    $\mathbf{v}^{(k)} \leftarrow$ GRID-SEARCH ($\mathbf{X}$)
    $\mathbf{X} \leftarrow$ projection of $\mathbf{X}$ onto the complement of $\mathbf{v}^{(k)}$
  end for
  Return subspace centered at $\hat{\mathbf{c}}\left\{\mathbf{x}^{(t)}\right\}$ with principal directions $\left\{\mathbf{v}^{(k)}\right\}_{k=1}^{K}$

To find the K-dimensional subspace $\left\{\mathbf{v}^{(k)} \;\middle|\; \forall j = 1, \ldots, K \; (\mathbf{v}^{(k)})^\top \mathbf{v}^{(j)} = \delta_{k,j}\right\}$ that maximizes the dispersion measure, the GRID-SEARCH is repeated K-times. After each repetition, the data is deflated to remove the dispersion captured by the last direction from the data. This process is detailed in Algorithm 5.2.

5.3.3 Robust Laplace Threshold

In addition to the robust PCA-GRID algorithm, I also design a robust estimate for its residual threshold that replaces of the Q-statistic described in Section 5.1.2. The use of the Q-statistic as a threshold by Lakhina et al. was implicitly motivated by an assumption of normally distributed residuals [Jackson and Mudholkar, 1979]. However, I found that the residuals for both the PCA and PCA-GRID subspaces were empirically non-normal leading me to conclude that the Q-statistic is a poor choice for a detection threshold. The non-normality of the residuals was also observed by Brauckhoff et al. [2009]. Instead, to account for the outliers and heavy-tailed behavior I observed from our method's residuals, I choose the threshold as the $1 - \beta$ quantile of a Laplace distribution fit with robust location and scale parameters. The alternative subspace-based anomaly detector, ANTIDOTE, is the combination of the PCA-GRID algorithm for normal-subspace estimation and the Laplace threshold to estimate the threshold for flagging anomalies.

As with the Q-statistic described in Section 5.1.2, I construct the Laplace threshold

$Q_{L,\beta}$ as the $1-\beta$ quantile of a parametric distribution fit to the residuals in the training data. However, instead of the normal distribution assumed by the Q-statistic, I use the quantiles of a Laplace distribution specified by a location parameter c and a scale parameter b. Critically, though, instead of using the mean and standard deviation, I robustly fit the distribution's parameters. I estimate c and b from the squared residuals $\left\{\left\|\ddot{\mathbf{x}}^{(t)}\right\|_2^2\right\}$ using robust consistent estimates $\hat{c}$ and $\hat{b}$ of location (median) and scale (MAD), respectively

$$\begin{aligned}\hat{c} &= \text{median}\left\{\left\|\ddot{\mathbf{x}}^{(t)}\right\|_2^2\right\} \\ \hat{b} &= \frac{1}{\sqrt{2}P^{-1}(0.75)}\,\text{MAD}\left\{\left\|\ddot{\mathbf{x}}^{(t)}\right\|_2^2\right\}\end{aligned}$$

where $P^{-1}(q)$ is the q^{th} quantile of the standard Laplace distribution. The Laplace quantile function has the form $P_{c,b}^{-1}(q) = c + b \cdot k_L(q)$ for the function $k_{Laplace}$ that is independent of the location and shape parameters of the distribution[6]. Thus, the Laplace threshold only depends linearly on the (robust) estimates $\hat{c}$ and $\hat{b}$ making the threshold itself robust. This form is also shared by the normal quantiles (differing only in its standard quantile function k_{Normal}), but because non-robust estimates for c and b are implicitly used by the Q-statistic, it is not robust. Further, by choosing the heavy-tailed Laplace distribution, the quantiles are more appropriate for the observed heavy-tailed behavior, but the robustness of this threshold is due to robust parameter estimation.

Empirically, the Laplace threshold also proved to be better suited for thresholding the residuals for ANTIDOTE than the Q-statistic. Figure 5.3(a) shows that both the Q-statistic and the Laplace threshold produce a reasonable threshold on the residuals of the PCA algorithm but, as seen in Figure 5.3(b), the Laplace threshold produces a reasonable threshold for the residuals of the PCA-GRID algorithm; the Q-statistic vastly underestimates the spread of the residuals. In the experiments described in the next section, the Laplace threshold is consistently more reliable than the Q-statistic.

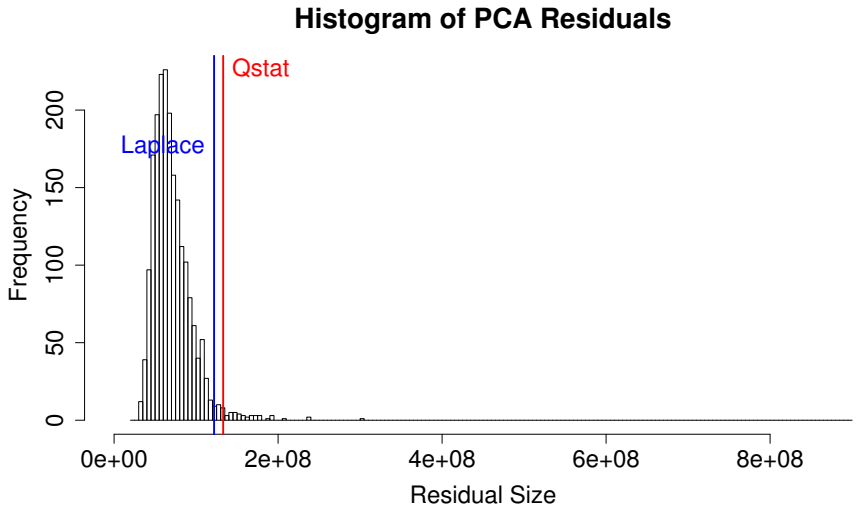
5.4 Empirical Evaluation

Here, I evaluate how the performance of PCA-based methods is affected by the poisoning strategies described in Section 5.2. I compare the original PCA-based detector and the alternative ANTIDOTE detector under these adversarial conditions using a variety of performance metrics.

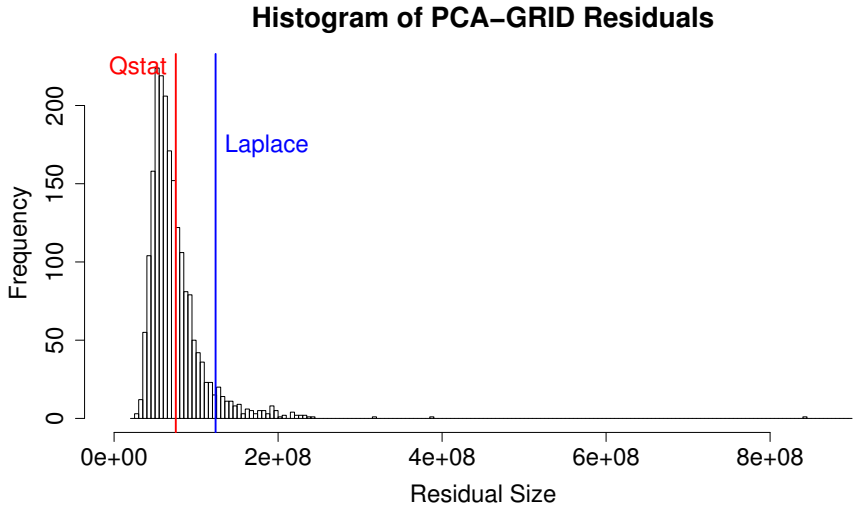
5.4.1 Setup

To assess the effect of poisoning, I test their performance for a variety of poisoning conditions. Here I describe the data used for that evaluation, the method used to test the detectors, and the different types of poisoning scenarios used in their evaluation.

[6]For the Laplace distribution, this function is given by $k_L(q) \triangleq \text{sign}\left(q-\frac{1}{2}\right)\cdot\ln\left(1-2\left|q-\frac{1}{2}\right|\right)$.



(a)



(b)

Figure 5.3: A comparison of the Q-statistic and the Laplace threshold for choosing an anomalous cutoff threshold for the residuals from an estimated subspace. **(a)** Histograms of the residuals for the original PCA algorithm and **(b)** of the PCA-GRID algorithm (the largest residual is excluded as an outlier). Red and blue vertical lines demarcate the threshold selected using the Q-statistic and the Laplace threshold, respectively. For the original PCA method, both methods choose nearly the same reasonable threshold to the right of the majority of the residuals. However, for the residuals of the PCA-GRID subspace, the Laplace threshold is reasonable whereas the Q-statistic is not; it would misclassify too much of the normal data to be an acceptable choice.

5.4.1.1 Traffic Data

The dataset I use for evaluation is OD flow data collected from the Abilene (Internet2 backbone) network to simulate attacks on PCA-based anomaly detection. This data was collected over an almost continuous 6 month period from March 1, 2004 through September 10, 2004 [Zhang, Ge, Greenberg, and Roughan, 2005]. Each week of data consists of 2016 measurements across all 144 network OD flows binned into 5 minute intervals. At the time of collection the network consisted of 12 PoPs and 15 inter-PoP links. 54 virtual links are present in the data corresponding to two directions for each inter-PoP link and an ingress and egress link for each PoP.

5.4.1.2 Validation

Although there are a total of 24 weeks of data in the dataset, these experiments are primarily based on the 20^{th} and 21^{st} weeks which span the period from August 7^{th}, 2004 to August 20^{th}, 2004. These weeks were selected because PCA achieved the lowest FNRs on these during testing and thus this data was most ideal for the detector. To evaluate a detector, it is trained on the 20^{th} week's traffic and tested on the data from the 21^{st} week during which DoS attacks are injected to measure how often the attacker can evade detection. To simulate the *Single-Training Period* attacks, the training traffic from week 21 is first poisoned by the attacker.

To evaluate the impact of poisoning on the original PCA-subspace method and ANTIDOTE in terms of their ability to detect DoS attacks, two consecutive weeks of data are used (again, the subsequent results use the 20^{th} and 21^{st} weeks)—the first for training and the second for testing. The poisoning occurs throughout the training phase, while the DoS attack occurs during the test week. An alternate evaluation method (described in detail below) is needed for the Boiling Frog scheme where training and poisoning occur over multiple weeks. The success of the poisoning strategies is measured by their impact on the subspace-based detector's false negative rate (FNR). The FNR is the ratio of the number of successful evasions to the total number of attacks (*i.e.*, the attacker's success rate is PCA's FNR rate). I also use Receiver Operating Characteristic (ROC) curves to visualize a detection method's trade-off between *true positive rate* (TPR) and false positive rate (FPR).

To compute the FNRs and FPRs, synthetic anomalies are generated according to the method of Lakhina et al. [2004b] and are injected into the Abilene data. While there are disadvantages to this method, such as the conservative assumption that a single volume size is anomalous for all flows, it is convenient for the purposes of relative comparison between PCA and Robust PCA, to measure relative effects of poisoning, and for consistency with prior studies. The training sets used in these experiments consist of week-long traffic traces, which is a sufficiently long time scale to capture weekday and weekend cyclic trends [Ringberg et al., 2007] and it is also the same time scale used in previous studies [Lakhina et al., 2004b]. Because the data is binned into five minute windows (corresponding to the reporting interval of SNMP), a decision about whether or not an anomaly occurred can be made at the end of each window; thus attacks can be detected within five minutes of their occurrence.

Unfortunately, computing the false positive rate of a detector is difficult since there may

be actual anomalous events in the Abilene data. To estimate the FPR, negative examples (benign OD flows) are generated as follows. The data is fit to an EWMA model that is intended to capture the main trends of the data with little noise. This model is used to select points in the Abilene flow's time series to use as negative examples. The actual data is then compared to the EWMA model; if the difference is small (not in the flow's top one percentile) for a particular flow at a particular time, $Q_{t,q}$, then the element $Q_{t,q}$ is labeled as *benign*. This process is repeated across all flows. The FPR of a detector is finally estimated based on the (false) alarms raised on the time slots that were deemed to be benign.

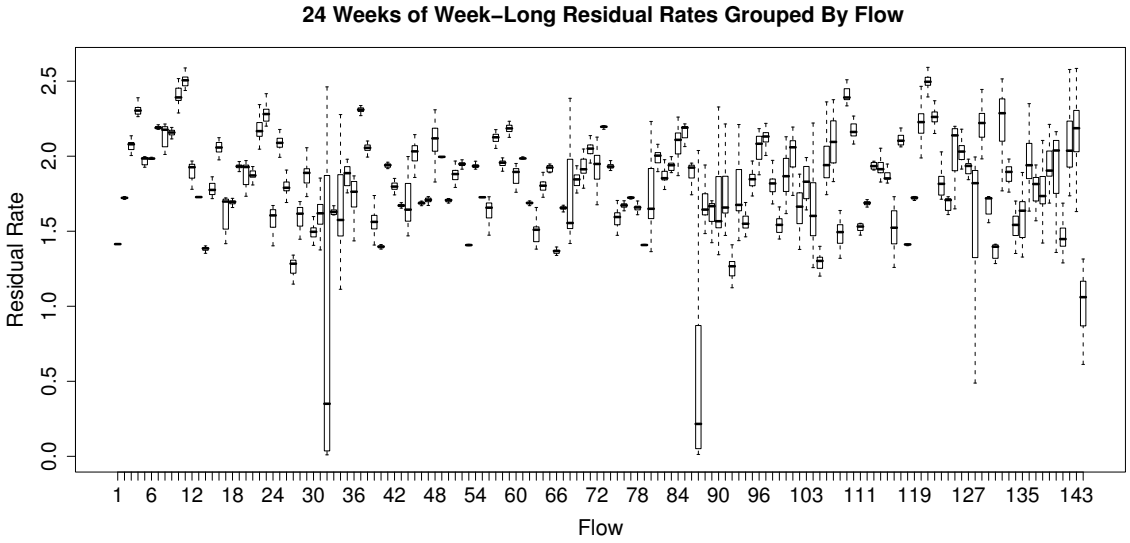
DoS attacks are simulated by selecting a target flow, q, and time window, t, and injecting a traffic spike along this target flow during the time window. Starting with the flow traffic matrix $\mathbf{Q}$ for the test week, a positive example (*i.e.*, an anomalous flow event) is generated by setting the q^{th} flow's volume at the t^{th} time window, $Q_{t,q}$, to be a large value known to correspond to an anomalous flow (replacing the original traffic volume in this time slot). This value was defined by Lakhina et al. [2004b] to be 1.5 times a cutoff of 8×10^7. After multiplying by the routing matrix $\mathbf{R}$, the link volume measurement at time t is anomalous. This process is repeated for each time t (*i.e.*, each five minute window) in the test week to generate 2016 anomalous samples for the q^{th} target flow.

A DoS attack is simulated along every flow at every time and the detector's alarms are recorded for each such attack. The FNR is estimated by averaging over all 144 flows and all 2016 time slots. When reporting the effect of an attack on traffic volumes, we first average over links within each flow then over flows. Furthermore, we generally report average volumes relative to the pre-attack average volumes. Thus, a single poisoning experiment was based on one week of poisoning with FNRs computed during the test week that includes 144×2016 samples coming from the different flows and time slots. Because the poisoning is deterministic in *Add-More-If-Bigger* this experiment was run once for that scheme. In contrast, for the *Random* poisoning scheme, we ran 20 independent repetitions of poisoning experiments data because the poisoning is random.

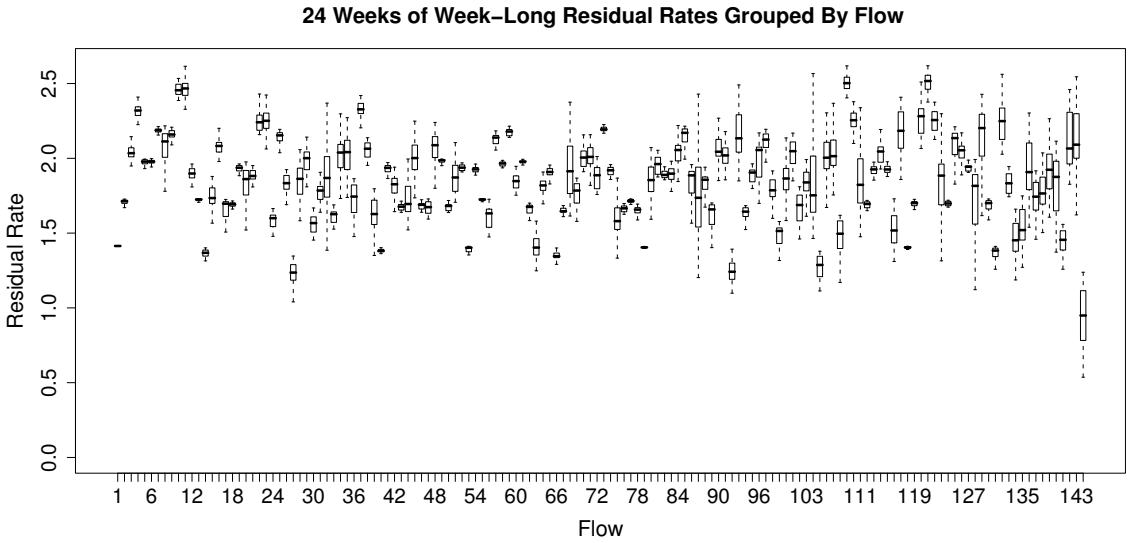
The squared prediction errors produced by the detection methods (based on the anomalous and normal examples from the test set) are used to produce ROC curves. By varying the method's threshold from $-\infty$ to ∞ a curve of possible $\langle FPR, TPR \rangle$ pairs is produced from the set of squared prediction errors; the Q-statistic and Laplace threshold, each correspond to one such point in ROC space. We adopt the Area Under Curve (AUC) statistic to directly compare ROC curves. The ideal detector has an AUC of 1, while the random predictor achieves an AUC of $\frac{1}{2}$.

5.4.2 Identifying Vulnerable Flows

There are two ways that a flow can be vulnerable. A flow is considered *vulnerable to DoS attack* (unpoisoned scenario) if a DoS attack along it is likely to be undetected when the resulting traffic data is projected onto the abnormal subspace. *Vulnerability to poisoning* means that if the flow is first poisoned, then the subsequent DoS attack is likely to undetected because the resulting projection in abnormal space is no longer significant. To examine the vulnerability of flows, I define the *residual rate* statistic, which measures the change in the size of the residual (*i.e.*, $\Delta \|\ddot{\mathbf{x}}\|_2$) caused by adding a single unit of traffic volume along a particular target flow. This statistic assesses how vulnerable a detector is to a DoS attack as it measures how rapidly the residual grows as the size of the DoS increases



(a) Residual Rates for PCA



(b) Residual Rates for PCA-Grid

Figure 5.4: Comparison of the original PCA subspace and PCA-Grid subspace in terms of their residual rates. Shown here are box plots of the 24 weekly residual rates for each flow to demonstrate the variation in residual rate for the two methods. **(a)** Distribution of the per-flow residual rates for the original PCA method and **(b)** for PCA-Grid. For PCA, flows 32 and 87 (the flows connecting Chicago and Los Angeles in Figure 5.1(b)) have consistently low residual rates making PCA susceptible to evasion along these flows. Both methods also have a moderate susceptibility along flow 144 (the ingress/egress link for Washington). Otherwise, PCA-Grid has overall high residual rates along all flows indicating little vulnerability to evasion.

and thus is an indicator of whether a large DoS attack will be undetected along a target flow. Injecting a unit volume along the q^{th} target flow causes an additive increase to the link measurement vector $\mathbf{R}_q$ and also increases the residual by

$$\nu\left(q;\ddot{\mathbf{P}}\right) \triangleq \left\|\ddot{\mathbf{P}}\mathbf{R}_q\right\|_2 \ .$$

The residual rate measures how well a flow aligns with normal subspace. If the flow aligns perfectly with the normal subspace, its residual rate will be 0 since changes along the directions of the subspace do not change the residual component of the traffic at all. More generally, a low residual rate indicates that (per unit of traffic sent) a DoS attack will not significantly impact the squared prediction error. Thus, for a detector to be effective, the residual rate must be high for most flows, otherwise the attacker will be able to execute large undetected DoS attacks.

By running PCA on each week of the Abilene data, I computed the residual rate of each flow for each week's model and estimated the spread in their residual rates. Figure 5.4 displays box plots of the residual rates for each flow over the 24 weeks of data. These plots show that when trained on uncontaminated data 99% of the flows have a median residual rate above 1.0; *i.e.*, for every unit of traffic added to any of these flows in a DoS attack, the residual component of the traffic increases by at least 1.0 unit and for many flows the increase is higher[7]. This result indicates that PCA trained on clean data is not vulnerable to DoS attacks on the majority of flows since each unit of traffic used in the attack increases the residual by at least one unit. However, PCA is very vulnerable to DoS attacks along flows 32 and 87 because their residual rates are small even without poisoning.

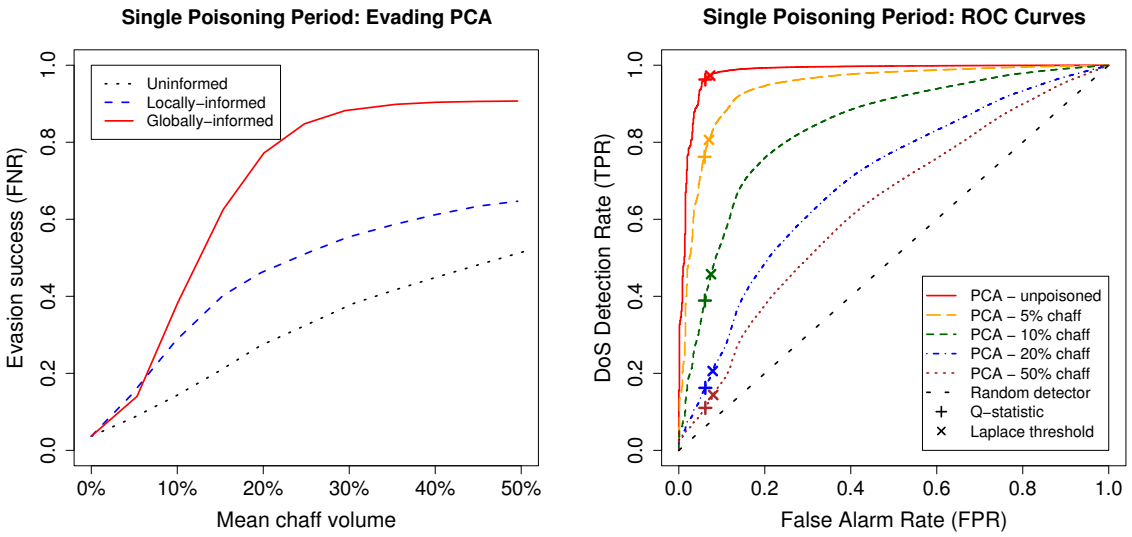
All of this is good news from the point of view of the attacker. Without poisoning, an attacker might only succeed if he were lucky enough to be attacking along the two highly vulnerable flows. However, after poisoning, it is clear that whatever the attack's target might be, the flow he chooses to attack, on average, is likely to be vulnerable.

5.4.3 Evaluation of Attacks

In this section, I present experimental validation that adversarial poisoning can have a significant detrimental impact on the PCA-based anomaly detector. I evaluate the effectiveness of the three data poisoning schemes from Section 5.2 for *Single-Training Period* attacks. During the testing week, the attacker launches a DoS attack in each 5 minute time window. The results of these attacks are displayed in Figure 5.5(a). Although the objective of these poisoning schemes is to add variance along the target flow, the mean of the target OD flow being poisoned increases as well, increasing the means of all links over which the OD flow traverses. The x-axis in Figure 5.5fig:week-long-fnr indicates the relative increase in the mean rate. The y-axis is the average FNR for that level of poisoning (*i.e.*, averaged over all OD flows).

As expected the increase in evasion success is smallest for the uninformed strategy, intermediate for the locally-informed scheme, and largest for the globally-informed poisoning scheme. A locally-informed attacker can use the *Add-More-If-Bigger* scheme to raise

[7]Many flows have residual rates well above 1 because these flow traverse many links and thus adding a single unit of traffic along the flow adds many units in link space. On average, flows in the Abilene dataset have 4.5 links per flow.



(a) Impact of Chaff on FNR

(b) Impact of Chaff on ROC curves

Figure 5.5: Effect of *Single-Training Period* poisoning attacks on the original PCA-based detector. **(a)** Evasion success of PCA versus relative chaff volume under *Single-Training Period* poisoning attacks using three chaff methods: uninformed (dotted black line) locally-informed (dashed blue line) and globally-informed (solid red line). **(b)**. Comparison of the ROC curves of PCA for different volumes of chaff (using *Add-More-If-Bigger* chaff). Also depicted are the points on the ROC curves selected by the Q-statistic and Laplace threshold, respectively.

his evasion success to 28% from the baseline FNR of 3.67% via a 10% average increase in the mean link rates due to chaff; *i.e.*, the attacker's rate of successful evasion increases nearly eight-fold from the rate of the unpoisoned PCA detector. With a *Globally-Informed* strategy, a 10% average increase in the mean link rates causes the unpoisoned FNR to increase by a factor of 10 to 38% success and eventually to over 90% FNR as the size of attack increases. The primary difference between the performance of the locally-informed and globally-informed attacker is intuitive to understand. Recall that the globally-informed attacker is privy to the traffic on all links for the entire training period while the locally-informed attacker only knows the traffic status of a single ingress link. Considering this information disparity, the locally-informed adversary is quite successful with only a small view of the network. An adversary is unlikely to be able to acquire, in practice, the capabilities used in the globally-informed poisoning attack. Moreover, adding 30% chaff, in order to obtain a 90% evasion success is dangerous in that the poisoning activity itself is likely to be detected. Therefore *Add-More-If-Bigger* offers a nice trade-off, from the adversary's point-of-view, in terms of poisoning effectiveness, and the attacker's capabilities and risks.

I also evaluate the PCA detection algorithm on both anomalous and normal data, as described in Section 5.4.1.2, to produce the Receiver Operating Characteristic (ROC) curves in Figure 5.5(b). I produce a series of ROC curves (as shown) by first training a PCA model on the unpoisoned data from the 20^{th} week and then training on data poisoned by progressively larger *Add-More-If-Bigger* attacks.

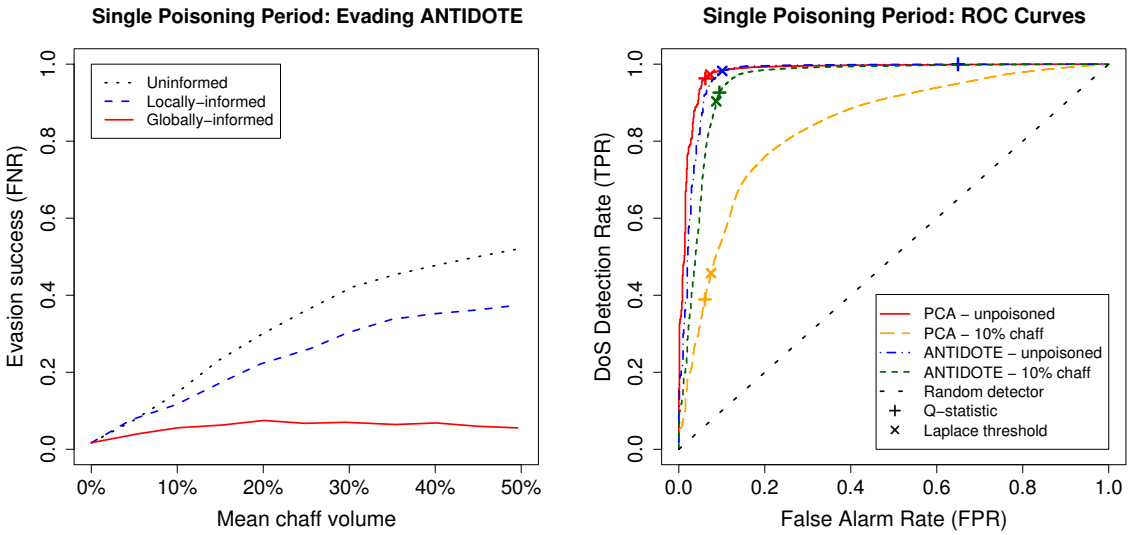
To validate PCA-based detection on poisoned training data, each flow is poisoned separately in different trials of the experiment as dictated by the threat model. Thus, for relative chaff volumes ranging from 5% to 50%, *Add-More-If-Bigger* chaff is added to each flow separately to construct 144 separate training sets and 144 corresponding ROC curves for the given level of poisoning. The poisoned curves in Figure 5.5(b) display the averages of these ROC curves; *i.e.*, the average TPR over the 144 flows for each FPR.

The sequence of ROC curves show that the *Add-More-If-Bigger* poisoning scheme creates an unacceptable trade-off between false positives and false negatives of the PCA detector: the detection and false alarm rates drop together rapidly as the level of chaff is increased. At 10% relative chaff volume performance degrades significantly from the ideal ROC curve (lines from $(0, 0)$ to $(0, 1)$ to $(1, 1)$) and at 20% the PCA's mean ROC curve is already close to that of a random detector (the $y = x$ line with an AUC of $\frac{1}{2}$).

5.4.4 Evaluation of Antidote

Here, I assess the effect poisoning attacks on ANTIDOTE performance during a single training period. As with the PCA-based detector, I evaluate the success of this detector with each of the different poisoning schemes and compute ROC curves using the *Add-More-If-Bigger* poisoning scheme to compare to the original PCA-subspace method.

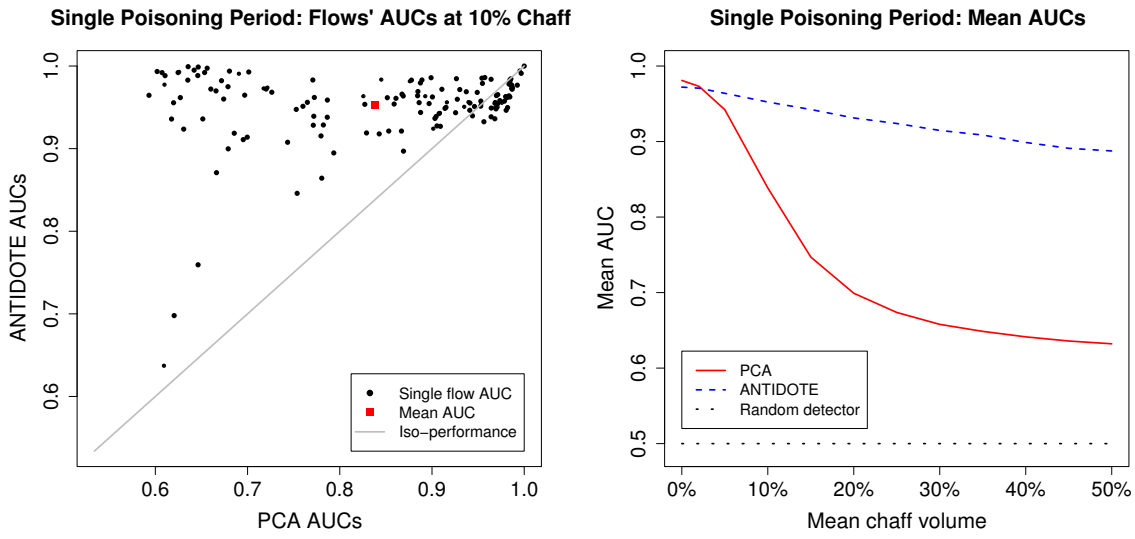
Figure 5.6(a) depicts ANTIDOTE's FNR for various levels of average poisoning that occur in a *Single-Training Period* attack compared to the results depicted in Figure 5.5(a) using the same metric for the original PCA detector. Comparing these results, the evasion success of the attack is dramatically reduced for ANTIDOTE. For any particular level of chaff, the evasion success rate of ANTIDOTE is approximately half that of the original PCA approach. Interestingly, the most effective poisoning scheme on PCA, *Globally-Informed*,



(a) Impact of Chaff on FNR

(b) Impact of Chaff on ROC curves

Figure 5.6: Effect of *Single-Training Period* poisoning attacks on the ANTIDOTE detector. **(a)** Evasion success of ANTIDOTE versus relative chaff volume under *Single-Training Period* poisoning attacks using three chaff methods: uninformed (dotted black line) locally-informed (dashed blue line) and globally-informed (solid red line). **(b)** Comparison of the ROC curves of ANTIDOTE and the original PCA detector when unpoisoned and under 10% chaff (using *Add-More-If-Bigger* chaff). The PCA detector and ANTIDOTE detector have similar performance when unpoisoned but PCA's ROC curve is significantly degraded with chaff whereas ANTIDOTE's is only slightly affected.



(a) (b)

Figure 5.7: Comparison of the original PCA detector in terms of the area under their (ROC) curves (AUCs). **(a)** The AUC for the PCA detector and the ANTIDOTE detector under 10% *Add-More-If-Bigger* chaff for each of the 144 target flows. Each point in this scatter plot is a single target flow; its x-coordinate is the AUC of PCA and its y-coordinate is the AUC of ANTIDOTE. Points above the line $y = x$ represent flows where ANTIDOTE has a better AUC than the PCA detector and those below $y = x$ represent flows for which PCA outperforms ANTIDOTE. The mean AUC for both methods is the red point. **(b)** The mean AUC of each detector versus the mean chaff level of an *Add-More-If-Bigger* poisoning attack for increasing levels of relative chaff. The methods compared are a random detector (dotted black line), the PCA detector (solid red line), and ANTIDOTE (dashed blue line).

is the least effective poisoning scheme against ANTIDOTE. The *Globally-Informed* scheme was designed in an approximately optimal fashion to circumvent PCA but for the alternative detector, *Globally-Informed* chaff is not optimized and empirically has little effect on PCA-GRID. For this detector, *Random* remains equally effective because constant shifts in a large subset of the data create a bimodality that is difficult for any subspace method to reconcile—since roughly half the data shifts by a constant amount, it is difficult to distinguish between the original and shifted subspaces. However, this effect is still small compared to the dramatic success of locally-informed and *Globally-Informed* chaff strategies against the original detector.

Since poisoning distorts the detector, it affects both the false negative and false positive rates. Figure 5.6(b) provides a comparison of the ROC curves for both ANTIDOTE and PCA when the training data is both unpoisoned and poisoned. For the poisoned training scenario, each point on the curve is the average over 144 poisoning scenarios in which the training data is poisoned along one of the 144 possible flows using the *Add-More-If-Bigger* strategy. While ANTIDOTE performs very similarly to PCA on unpoisoned training data, PCA's performance is significantly degraded by poisoning while ANTIDOTE remains relatively unaffected. With a moderate mean chaff volume of 10%, ANTIDOTE's average ROC curve remains close to optimal while PCA's curve considerably shifts towards the $y = x$ curve of the random detector. This means that under a moderate level of poisoning, PCA cannot achieve a reasonable trade-off between false positives and false negatives while ANTIDOTE retains a good operating point for these two common performance measures. *In summary, in terms of false positives and false negatives,* ANTIDOTE *incurs insignificant performance shifts when no poisoning occurs, but is resilient against poisoning and provides enormous performance gains compared to PCA when poisoning attacks do occur.*

Given Figures 5.6(a) and 5.6(b) alone, it is conceivable that ANTIDOTE outperforms PCA only on average, and not on all flows targeted for poisoning. In place of plotting all 144 poisoned ROC curves, Figure 5.7(a) compares the AUCs for the two detection methods under 10% chaff. Not only is average performance much better for robust PCA, but it in fact outperforms PCA on most flows and by a decidedly large amount. Although PCA indeed performs slightly better for some flows, in these cases both methods have excellent detection performance (because their AUCs are close to 1), and hence the distinction between the two is insignificant for those specific flows.

Figure 5.7(b) plots the mean AUC (averaged from the 144 ROC curves' AUCs where flows are poisoned separately) achieved by the detectors for an increasing level of poisoning. ANTIDOTE behaves comparably to albeit slightly worse than PCA under no chaff conditions, yet its performance remains relatively stable as the amount of contamination increases while PCA's performance rapidly degrades. In fact, with as little as 5% poisoning, ANTIDOTE already exceeds the performance of PCA and the gap only widens with increasing contamination. As PCA's performance drops, it approaches a random detector (equivalently, $AUC = \frac{1}{2}$), for amounts of poisoning exceeding 20%. As these experiments demonstrate, ANTIDOTE is an effective defense and dramatically outperforms a solution that was not designed to be robust. This is strong evidence that the robust techniques are a promising instrument for designing machine learning algorithms used in security-sensitive domains.

5.4.5 Empirical Evaluation of *Boiling Frog*

5.4.5.1 Experimental Methodology for Episodic Poisoning

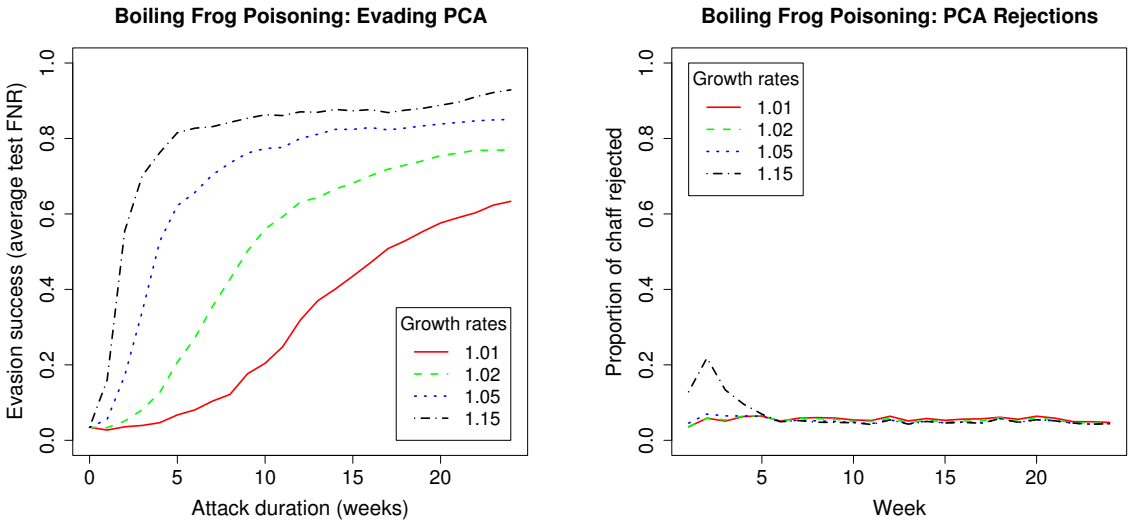
To test the *Boiling Frog* attacks, several weeks of traffic data are simulated using a generative model inspired by Lakhina, Crovella, and Diot [2004b]. These simulations produce multiple weeks of data generated from a stationary distribution. While such data is unrealistic in practice, stationary data is the ideal dataset for PCA to produce a reliable detector. Anomaly detection under non-stationary conditions is more difficult due to the learner's inability to distinguish between benign data drift and anomalous conditions. By showing that PCA is susceptibility to episodic poisoning even in this stationary case, these experiments suggest that the method can also be compromised in more realistic settings. Further, the six month Abilene dataset of Zhang et al. [2005] proved to be too non-stationary for PCA to consistently operate well from one week to the next—PCA often performed poorly even without poisoning. It is unclear whether the non-stationarity observed in this data is prevalent in general or whether it is an artifact of the dataset, but nonetheless, these experiments show PCA is susceptible to poisoning even when the underlying data is well-behaved.

To synthesize a stationary multi-week dataset of OD flow traffic matrices, a three step generative procedure is used to model each OD flow separately. First the underlying daily cycle of the q^{th} OD flow's time series is modeled by a sinusoidal approximation. Then the times at which the flow is experiencing an anomaly are modeled by a Binomial arrival process with inter-arrival times distributed according to the geometric distribution. Finally Gaussian white noise is added to the base sinusoidal model during times of benign OD flow traffic; and exponential traffic is added to the base model during times of anomalous traffic.

In the first step, the underlying cyclic trends are captured by fitting the coefficients for Fourier basis functions. Following the model proposed by Lakhina et al. [2004b], the basis functions are sinusoids of periods of 7, 5 and 3 days, and 24, 12, 6, 3 and 1.5 hours, as well as a constant function. For each OD flow, the Fourier coefficients are estimated by projecting the flow onto this basis. The portion of the traffic modeled by this Fourier forecaster is removed and the remaining residual traffic is modeled with two processes—a zero-mean Gaussian noise process captures short-term benign traffic variance and an exponential distribution is used to model non-malicious volume anomalies.

In the second step, one of the two noise processes is selected for each time interval. After computing the Fourier model's residuals (the difference between the observed and predicted traffic) the smallest negative residual value $-m$ is recorded. We assume that residuals in the interval $[-m, m]$ correspond to benign traffic and that residuals exceeding m correspond to traffic anomalies (this is an approximation but it works reasonably well for most OD flows). Periods of benign variation and anomalies are then modeled separately since these effects behave quite differently. Upon classifying residual traffic as benign or anomalous, anomaly arrival times are modeled as a Bernoulli arrival process and the inter-anomaly arrival times are geometrically distributed. Further, since we consider only spatial PCA methods, the temporal placement of anomalies is unimportant.

In the third and final step, the parameters for the two residual traffic volume and the inter-anomaly arrival processes are inferred from the residual traffic using the maximum likelihood estimates of the Gaussian's variance and exponential and geometric rates respec-



(a) Effect of *Boiling Frog* on PCA

(b) Rejection of Chaff by PCA

Figure 5.8: Effect of *Boiling Frog* poisoning attacks on the original PCA-subspace detector (see Figure 5.9 for comparison with the PCA-based detector). **(a)** Evasion success of PCA under *Boiling Frog* poisoning attacks in terms of the average FNR after each successive week of poisoning for four different poisoning schedules (*i.e.*, a weekly geometric increase in the size of the poisoning by factors 1.01, 1.02, 1.05, and 1.15 respectively). More aggressive schedules (*e.g.*, growth rates of 1.05 and 1.15) significantly increase the FNR within a few weeks while less aggressive schedules take many weeks to achieve the same result but are more stealthy in doing so. **(b)** Weekly chaff rejection rates by the PCA-based detector for the *Boiling Frog* poisoning attacks from Figure (a). The detector only detects a significant amount of the chaff during the first weeks of the most aggressive schedule (growth rate of 1.15); subsequently, the detector is too contaminated to accurately detect the chaff.

tively. Positive goodness-of-fit results (Q-Q plots not shown) have been obtained for small, medium and large flows.

In the synthesis, all link volumes are constrained to respect the link capacities in the Abilene network: 10gbps for all but one link that operates at one fourth of this rate. We also cap chaff that would cause traffic to exceed the link capacities.

5.4.5.2 Effect of Episodic Poisoning on the PCA Detector

I now evaluate the effectiveness of the *Boiling Frog* strategy, that contaminates the training data over multiple training periods. Figure 5.8(a) plots the FNRs against the poisoning duration for the PCA detector for four different *poisoning schedules* with growth rates of 1.01, 1.02, 1.05 and 1.15 respectively. The schedule's growth rate corresponds to the rate of increase in the attacked links' average traffic from week to week. The attack strength parameter θ (*cf.*, Section 5.2) is selected to achieve this goal. We see that the FNR dramatically increases for all four schedules as the poison duration increases. With a 15% growth

rate the FNR is increased from 3.67% to more than 70% over three weeks of poisoning; even with a 5% growth rate the FNR is increased to 50% over 3 weeks. Thus *Boiling Frog* attacks are effective even when the amount of poisoned data increases rather slowly. Further, in comparing Figure 5.5(a) for *Single-Training Period* to Figure 5.8(a), the success of *Boiling Frog* attacks becomes clear. For the *Single-Training Period* attack, to raise the FNR to 50%, an immediate increase in mean traffic of roughly 18% is required, whereas in the *Boiling Frog* attack the same result can be achieved with only a 5% average traffic increase spread across three weeks.

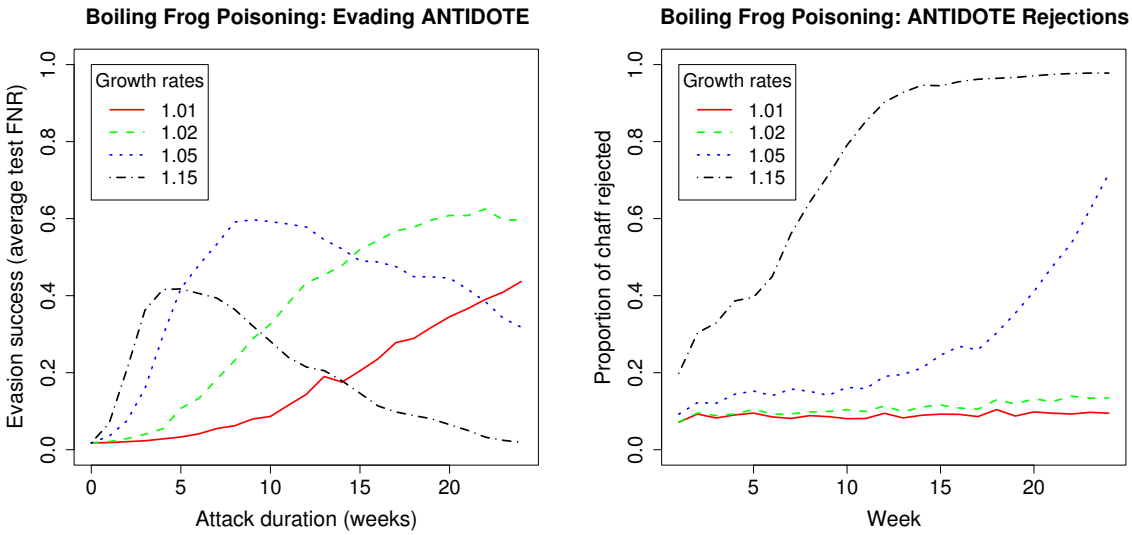
Recall that the two methods are retrained every week using the data collected from the previous week. However, the data from the previous week is also filtered by the detector itself and any time window flagged as anomalous, the training data is thrown out. Figure 5.8(b) shows the proportion of chaff rejected each week by PCA (*chaff rejection rate*) for the *Boiling Frog* strategy. The three slower schedules enjoy a relatively small constant rejection rate close to 5%. The 15% schedule begins with a relatively high rejection rate, but after a month sufficient amounts of poisoned traffic mis-train PCA after which point the rates drop to the level of the slower schedules. Thus, the *Boiling Frog* strategy with a moderate growth rate of 2–5% can significantly poison PCA, dramatically increasing its FNR while still going unnoticed by the detector.

5.4.5.3 Effect of Episodic Poisoning on Antidote

I now evaluate effectiveness of ANTIDOTE against the *Boiling Frog* strategy that occurs over multiple successive training periods. Figure 5.9(a) shows the FNRs for ANTIDOTE with the four different poisoning schedules (recall from Section 5.4.5.2 that each is the weekly growth factor for the increase in size of a *Add-More-If-Bigger* poisoning strategy). First, for the two most stealthy poisoning strategies (1.01 and 1.02), ANTIDOTE shows remarkable resistance in that the evasion success increases very slowly, *e.g.*, after ten training periods it is still below 20% evasion success. This is in stark contrast to PCA (see Figure 5.8(a)); for example, after ten weeks the evasion success against PCA exceeds 50% for the 1.02 poisoning growth rate scenario.

Second, under PCA the evasion success consistently increases with each additional week. However, with ANTIDOTE, the evasion success of these more aggressive schedules actually decreases after several weeks. The reason is that as the chaff levels rise, ANTIDOTE increasingly is able to identify the chaff as abnormal and then reject enough of it from the subsequent training data that the poisoning strategy loses its effectiveness.

Figure 5.9(b) shows the proportion of chaff rejected by ANTIDOTE under episodic poisoning. The two slower schedules almost have a constant rejection rate close to 9% (which is higher than PCA's rejection rate of around 5%). For the more aggressive growth schedules (5% and 15%), however, ANTIDOTE rejects an increasing amount of the poison data. This reflects a good target behavior for any robust detector—to reject more training data as the contamination grows. *Overall, these experiments provide empirical evidence that the combination of techniques used by* ANTIDOTE*, namely a subspace-based detector designed with robust subspace estimator combined with a Laplace-based cutoff threshold, maintains a good balance between false negative and false positive rates throughout a variety of poisoning scenarios (different amounts of poisoning, on different OD flows, and on different time horizons) and thus provides a resilient alternative to the original PCA-based detector.*



(a) Effect of *Boiling Frog* on ANTIDOTE

(b) Rejection of Chaff by ANTIDOTE

Figure 5.9: Effect of *Boiling Frog* poisoning attacks on the ANTIDOTE detector (see Figure 5.8 for comparison with the PCA-based detector). **(a)** Evasion success of ANTIDOTE under *Boiling Frog* poisoning attacks in terms of the average FNR after each successive week of poisoning for four different poisoning schedules (*i.e.*, a weekly geometric increase in the size of the poisoning by factors 1.01, 1.02, 1.05, and 1.15 respectively). Unlike the weekly FNRs for the *Boiling Frog* poisoning in Figure 5.8(a), the more aggressive schedules (*e.g.*, growth rates of 1.05 and 1.15) reach their peak FNR after only a few weeks of poisoning after which their effect declines (as the detector successfully rejects increasing amounts of chaff). The less aggressive schedules (with growth rates of 1.01 and 1.02) still have gradually increasing FNRs, but also seem to eventually plateau. **(b)** Weekly chaff rejection rates by the ANTIDOTE detector for the *Boiling Frog* poisoning attacks from Figure (a). Unlike PCA (see Figure 5.8(b)), ANTIDOTE rejects increasingly more chaff from the *Boiling Frog* attack. For all poisoning schedules, ANTIDOTE has a higher baseline rejection rate (around 10%) than the PCA detector (around 5%) and it rejects most of the chaff from aggressive schedules within a few weeks. This suggests that, unlike PCA, ANTIDOTE is not progressively poisoned by increasing week-to-week chaff volumes.

5.5 Summary

To subvert the PCA-based detector proposed by Lakhina et al. [2004b], I studied *Causative Integrity* attacks that poison the training data by adding malicious chaff; *i.e.*, spurious traffic sent across the network by compromised nodes that reside within it. This chaff is designed to interfere with PCA's subspace estimation procedure. Based on a relaxed objection function, I demonstrated how an adversary can approximate optimal noise using a global view of the traffic patterns in the network. Empirically, I found that by increasing the mean link rate by 10% with *Globally-Informed* chaff traffic, the FNR increased from 3.67% to 38%—a ten-fold increase in misclassification of DoS attacks. Similarly, by only using local link information the attacker is able to mount a more realistic *Add-More-If-Bigger* attack. For this attack, increasing the mean link rate by 10% with *Add-More-If-Bigger* chaff traffic, the FNR increased from 3.67% to 28%—an eight-fold increase in misclassification of DoS attacks. These attacks demonstrate that with sufficient information about network patterns, attacks can mount attacks against the PCA detector that severely compromises its ability to detect future DoS attacks traversing the networking it is monitoring.

I also demonstrated that an alternative robust method for subspace estimation could be used instead to make the resulting DoS detector less susceptible to poisoning attacks. The alternative detector was constructed using a subspace method for robust PCA developed by Croux et al. and a more robust method for estimating the residual cutoff threshold. The resulting ANTIDOTE detector is impacted by poisoning but its performance degrades more gracefully. Under non-poisoned traffic, ANTIDOTE performs nearly as well as PCA, but for all levels of contamination using *Add-More-If-Bigger* chaff traffic, the misclassification rate of ANTIDOTE is approximately half the FNR of the PCA-based solution. Moreover, the average performance of ANTIDOTE is much better than the original detector; it outperforms ordinary PCA for more flows and by a large amount. For multi-week *Boiling Frog* attacks, ANTIDOTE also outperformed PCA and would catch progressively more attack traffic in each subsequent week.

5.5.1 Future Work

Several important questions about subspace detection methods remain unanswered. While I have demonstrated that ANTIDOTE is resilient to poisoning attacks, it is not yet known if there are alternative poisoning schemes that significantly reduce ANTIDOTE's detection performance. Because ANTIDOTE is founded on robust estimators, it is unlikely that there is a poisoning strategy that completely degrades its performance. However, to better understand the limits of attacks and defenses, it is imperative to continue investigating worst-case attacks against the next generation of defenders; in this case, ANTIDOTE.

> QUESTION 5.1 What are the worst-case poisoning attacks against the ANTIDOTE-subspace detector for large-volume network anomalies? What are game-theoretic equilibrium strategies for the attacker and defender in this setting? How does ANTIDOTE's performance compare to these strategies?

There are also several other approaches for developing effective anomaly detectors for large volume anomalies [*e.g.*, Brauckhoff et al., 2009]. To compare these alternatives to ANTIDOTE, one must first identify their vulnerabilities and assess their performance when

under attack. More importantly though, I think detectors could be substantially improved by combining them together.

> QUESTION 5.2 Can subspace-based detection approaches be adapted to incorporate the alternative approaches? Can they find both temporal and spatial correlations and use both to detect anomalies? Can subspace-based approaches be adapted to incorporate domain-specific information such as the topology of the network?

Developing the next generation of network anomaly detectors is a critical task that perhaps can incorporate several of the themes I promote in this dissertation to create secure learners.

Part II

Partial Reverse-Engineering of Classifiers through Near-Optimal Evasion

Chapter 6

Near-Optimal Evasion of Classifiers

In this chapter, I explore a theoretical model for quantifying the difficulty of *Exploratory* attacks against a trained classifier. Unlike the previous work, since the classifier has already been trained, the adversary can no longer exploit vulnerabilities in the learning algorithm to mis-train the classifier as I demonstrated in the first part of this dissertation. Instead, the adversary must exploit vulnerabilities that the classifier accidently acquired from training on benign data (or at least data not controlled by the adversary in question). Most non-trivial classification tasks will lead to some form of vulnerability in the classifier. All known detection techniques are susceptible to blind spots (*i.e.*, classes of miscreant activity that fail to be detected), but simply knowing that they exist is insufficient. The principle question is how difficult it is for an adversary to discover a blind spot that is most advantageous for the adversary. In this chapter, I explore a framework for quantifying how difficult it is for the adversary to search for this type of vulnerability in a classifier.

At first, it may appear that the ultimate goal of these *Exploratory* attacks is to reverse-engineer the learned parameters, internal state, or the entire boundary of a classifier to discover its blind spots. However, in this work, I adopt a more refined strategy; I demonstrate successful *Exploratory* attacks that only *partially* reverse-engineer the classifier. My techniques find blind spots using only a *small* number of queries and yield near-optimal strategies for the adversary. They discover data points that the classifier will classify as benign and that are close to the adversary's desired attack instance.

While learning algorithms allow the detection algorithm to adapt over time, real-world constraints on the learning algorithm typically allow an adversary to programmatically find blind spots in the classifier. I consider how an adversary can systematically discover blind spots by querying the filter to find a low cost (for some cost function) instance that evades the filter. Consider, for example, a spammer who wishes to minimally modify a spam message so it is not classified as a spam (here cost is a measure of how much the spam must be modified). By observing the responses of the spam detector[1], the spammer can search for a modification while using few queries.

The problem of near-optimal evasion (*i.e.*, finding a low cost negative instance with few queries) was introduced by Lowd and Meek [2005b]. I continue studying this problem by

[1]There are a variety of domain specific mechanisms an adversary can use to observe the classifier's response to a query; *e.g.*, the spam filter of a public email system can be observed by creating a *test* account on that system and sending the queries to that account. In this paper, I assume the filter is queryable.

generalizing it to the family of convex-inducing classifiers—classifiers that partition their instance space into two sets one of which is convex. The family of convex-inducing classifiers is a particularly important and natural set of classifiers to examine which includes the family of linear classifiers studied by Lowd and Meek as well as anomaly detection classifiers using bounded PCA [Lakhina et al., 2004b], anomaly detection algorithms that use hyper-sphere boundaries Bishop [2006], one-class classifiers that predict anomalies by thresholding the log-likelihood of a log-concave (or uni-modal) density function, and quadratic classifiers of the form $\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + c \geq 0$ if $\mathbf{A}$ is semidefinite [see Boyd and Vandenberghe, 2004, Chapter 3], to name a few. The family of convex-inducing classifiers also includes more complicated bodies such as the countable intersection of halfspaces, cones, or balls.

I further show that near-optimal evasion does not require complete reverse-engineering of the classifier's internal state or decision boundary, but instead, only partial knowledge about its general structure. The algorithm of Lowd and Meek [2005b] for evading linear classifiers reverse-engineers the decision boundary by estimating the parameters of their separating hyperplane. The algorithms I present for evading convex-inducing classifiers do not require fully estimating the classifier's boundary (which is hard in the case of general convex bodies; see Rademacher and Goyal, 2009) or the classifier's parameters (internal state). Instead, these algorithms directly search for a minimal cost-evading instance. These search algorithms require only polynomial-many queries, with one algorithm solving the linear case with better query complexity than the previously-published reverse-engineering technique. Finally, I also extend near-optimal evasion to general ℓ_p costs. I show that the algorithms for ℓ_1 costs can also be extended to near-optimal evasion on ℓ_p costs, but are generally not efficient. However, in the cases when these algorithms are not efficient, I show that there is no efficient query-based algorithm.

The results presented in this chapter were previously published as the report *Query Strategies for Evading convex-inducing classifiers* [Nelson et al., 2010b] that extends an earlier paper I published with my colleagues [Nelson et al., 2010a]. Also, many of the open questions suggested at the end of this chapter first appeared in *Classifier Evasion: Models and Open Problems* [Nelson et al., 2010c]. The rest of this chapter is organized as follows. I first present an overview of the prior work most closely related to the near-optimal evasion problem in the remainder of this section (see Chapter 3 for additional related work). In Section 6.1 I formalize the near-optimal evasion problem, and review Lowd and Meek's definitions and results. I present algorithms for evasion that are near-optimal under weighted ℓ_1 costs in Section 6.2 and I provide results for minimizing general ℓ_p costs in Section 6.3.

Related Work Lowd and Meek [2005b] first explored near-optimal evasion, and developed a method that reverse-engineered linear classifiers as discussed in Chapter 3.4.2.4 and 3.4.4. The theory I present here generalizes their original results and provides three significant improvements:

- This analysis considers a more general family of classifiers: the family of convex-inducing classifiers that partition the space of instances into two sets one of which is convex. This family subsumes the family of linear classifiers considered by Lowd and Meek.
- The approach I present does not fully estimate the classifier's decision boundary

(which is generally hard for arbitrary convex bodies Rademacher and Goyal, 2009) or reverse-engineer the classifier's state. Instead, the algorithms search directly for an instance that the classifier labels as negative that is close to the desired attack instance; *i.e.*, an evading instance of near-minimal cost.

- Despite being able to evade a more general family of classifiers, these algorithms still only use a limited number of queries: they require only a number of queries polynomial in the dimension of the instance space and the desired accuracy of the approximation. Moreover, the K-step MultiLineSearch (Algorithm 6.4) solves the linear case with asymptotically fewer queries than the previously-published reverse-engineering technique for this case.

Further, as summarized in Chapter 3.4.2.4, Dalvi et al., Brückner and Scheffer, and Kantarcioglu et al. studied cost-sensitive game theoretic approaches to preemptively patch a classifier's blind spots and developed techniques for computing an equilibrium for their games. This work is complementary to query-based evasion problems; the near-optimal evasion problem studies how an adversary can use queries to find blind spots of a classifier that is unknown but queryable whereas their game-theoretic approaches assume the adversary knows the classifier and can optimize their evasion accordingly at each step of an iterated game. Thus, the near-optimal evasion setting studies how difficult it is for an adversary to optimize their evasion strategy only by querying and cost-sensitive game-theoretic learning studies how the adversary and learner can optimally play and adapt in the evasion game given knowledge of each other: two separate aspects of evasion.

A number of authors also studied evading sequence-based IDSs as discussed in Chapter 3.4.2.2 [see Tan et al., 2002, 2003, Wagner and Soto, 2002]. In exploring *mimicry attacks*, these authors used offline analysis of the IDSs to construct their modifications; by contrast, the adversary in near-optimal evasion constructs optimized modifications designed by querying the classifier.

The field of active learning also studies a form of query based optimization [Schohn and Cohn, 2000, *e.g.*, see]. While both active learning and near-optimal evasion explore optimal querying strategies, the objectives for these two settings are quite different (see Chapter 6.1.2 for further discussion on these differences).

6.1 Characterizing Near-Optimal Evasion

I begin by introducing the assumptions made for this problem. First, I assume that feature space $\mathcal{X}$ for the learner is a real-valued D-dimensional Euclidean space; *i.e.*, $\mathcal{X} = \Re^D$. (Lowd and Meek also consider integer and Boolean valued instance spaces and provide interesting results for several classes of Boolean-valued learners, but these spaces are not compatible with the family of convex-inducing classifiers I study in this chapter.) I assume that the feature space representation is known to the adversary and there are no restrictions on the adversary's queries; *i.e.*, any point in feature space $\mathcal{X}$ can be queried by the adversary. These assumptions may not be true in every real-world setting, but allow us to consider a worst-case adversary.

As in Chapter 2.2.4, I assume the target classifier f is a member of a family of classifiers $\mathcal{F}$—the adversary does not know f but knows the family $\mathcal{F}$. (This knowledge is congruous

with the security assumption that the adversary knows the learning algorithm but not the training set or parameters used to tune the learner.) I also restrict my attention to binary classifiers and use $\mathcal{Y} = \{'-', '+'\}$. I assume the adversary's attack will be against a fixed f so the learning method and the training data used to select f are irrelevant for this problem. Further, I assume $f \in \mathcal{F}$ is deterministic and so it partitions $\mathcal{X}$ into 2 sets—the positive class $\mathcal{X}_f^+ = \{\mathbf{x} \in \mathcal{X} \mid f(\mathbf{x}) = '+'\}$ and the negative class $\mathcal{X}_f^- = \{\mathbf{x} \in \mathcal{X} \mid f(\mathbf{x}) = '-'\}$. As before, I take the negative set to be *normal* instances where the sought after blind spots reside. I assume that the adversary is aware of at least one instance in each class, $\mathbf{x}^- \in \mathcal{X}_f^-$ and $\mathbf{x}^A \in \mathcal{X}_f^+$, and can observe the class for any $\mathbf{x}$ by issuing a *membership query*: $f(\mathbf{x})$.

6.1.1 Adversarial Cost

I assume the adversary has a notion of utility over the instance space which I quantify with a cost function $A : \mathcal{X} \mapsto \Re^{0+}$. The adversary wishes to optimize A over the negative class, $\mathcal{X}_f^-$; *e.g.*, a spammer wants to send spam that will be classified as normal email ('−') rather than as spam ('+'). I assume this cost function is a distance to some instance $\mathbf{x}^A \in \mathcal{X}_f^+$ that is most desirable to the adversary; *e.g.*, for a spammer this could be the string edit distance required to change $\mathbf{x}^A$ to a different message. I focus on the general class of weighted ℓ_p ($0 < p \leq \infty$) cost functions relative to $\mathbf{x}^A$ defined in terms of the ℓ_p norm $\|\cdot\|_p$ as:

$$A_p^{(\mathbf{c})}(\mathbf{x}) = \left\|\mathbf{c} \odot \left(\mathbf{x} - \mathbf{x}^A\right)\right\|_p = \left(\sum_{d=1}^{D} c_d^p \left|x_d - x_d^A\right|^p\right)^{1/p} , \tag{6.1}$$

where $0 < c_d < \infty$ is the relative cost the adversary associates with the d^{th} feature. In Section 6.2.1.3, I also consider the special cases when some features have $c_d = 0$ (the adversary doesn't care about the d^{th} feature) or $c_d = \infty$ (the adversary requires the d^{th} feature to match x_d^A), but otherwise, the weights are on the interval $(0, \infty)$. Weighted ℓ_1 costs are particularly appropriate for many adversarial problems since costs are assessed based on the degree to which a feature is altered and the adversary typically is interested in some features more than others. The ℓ_1-norm is a natural measure of edit distance for email spam, while larger weights can model tokens that are more costly to remove (*e.g.*, a payload URL). As with Lowd and Meek, I focus primarily on weighted ℓ_1 costs in Chapter 6.2 then explore general ℓ_p costs in Chapter 6.3. I use $\mathbb{B}^C(A)$ to denote the C-cost ball (or sublevel set) with cost no more than C; *i.e.*, $\mathbb{B}^C(A) = \{\mathbf{x} \in \mathcal{X} \mid A(\mathbf{x}) \leq C\}$. For instance, $\mathbb{B}^C(A_1)$ is the set of instances that do not exceed an ℓ_1 cost of C from the target $\mathbf{x}^A$.

Lowd and Meek [2005b] define *minimal adversarial cost* (MAC) of a classifier f to be the value

$$MAC(f, A) \triangleq \inf_{\mathbf{x} \in \mathcal{X}_f^-} [A(\mathbf{x})] \ ; \tag{6.2}$$

i.e., the greatest lower bound on the cost obtained by any negative instance. They further define a data point to be an ϵ-approximate *instance of minimal adversarial cost (ϵ-IMAC)* if it is a negative instance with a cost no more than a factor $(1+\epsilon)$ of the MAC; *i.e.*, every ϵ-$IMAC$ is a member of the set[2]

$$\epsilon\text{-}IMAC(f, A) \triangleq \left\{\mathbf{x} \in \mathcal{X}_f^- \;\middle|\; A(\mathbf{x}) \leq (1+\epsilon) \cdot \text{MAC}(f, A)\right\} . \tag{6.3}$$

[2]I use the term ϵ-$IMAC$ to refer both to this set and members of it. The usage will be clear from the context.

Alternatively, this set can be characterized as the intersection of the negative class and the ball of A of costs within a factor $(1+\epsilon)$ of $MAC(f, A)$ (*i.e.*, ϵ-$IMAC(f, A) = \mathcal{X}_f^- \cap \mathbb{B}^{(1+\epsilon)\cdot MAC}(A)$); a fact I exploit in Chapter 6.2.2. The adversary's goal is to find an ϵ-$IMAC$ efficiently, while issuing as few queries as possible. In the next section, I introduce formal notions to quantify how effectively an adversary can achieve this objective.

6.1.2 Near-Optimal Evasion

Lowd and Meek [2005b] introduce the concept of *adversarial classifier reverse engineering (ACRE) learnability* to quantify the difficulty of find an ϵ-$IMAC$ instance for a particular family of classifiers, $\mathcal{F}$, and a family of adversarial costs, $\mathcal{A}$. Using my notation, their definition of $ACRE$ ϵ-learnable is

> A set of classifiers $\mathcal{F}$ is $ACRE$ ϵ-learnable under a set of cost functions $\mathcal{A}$ if an algorithm exists such that for all $f \in \mathcal{F}$ and $A \in \mathcal{A}$, it can find a $\mathbf{x} \in \epsilon$-$IMAC(f, A)$ using only polynomially many membership queries in D, the encoded size of f, and the encoded size of $\mathbf{x}^+$ and $\mathbf{x}^-$.

In generalizing their result, I use a slightly altered definition of query complexity. First, to quantify query complexity, I only use the dimension D and the number of steps L_ϵ required by a unidirectional binary search to narrow the gap to within a factor $1+\epsilon$, the desired accuracy[3]. Second, I assume the adversary only has two initial points $\mathbf{x}^- \in \mathcal{X}_f^-$ and $\mathbf{x}^A \in \mathcal{X}_f^+$ (the original setting required a third $\mathbf{x}^+ \in \mathcal{X}_f^+$); this yields simpler search procedures[4]. Finally, my algorithms do not reverse engineer so ACRE would be a misnomer. Instead, I call the overall problem near-optimal evasion and replace $ACRE$ ϵ-learnable with the following definition of ϵ-$IMAC$ searchable.

> A family of classifiers $\mathcal{F}$ is ϵ-$IMAC$ *searchable* under a family of cost functions $\mathcal{A}$ if for all $f \in \mathcal{F}$ and $A \in \mathcal{A}$, there is an algorithm that finds $\mathbf{x} \in \epsilon$-$IMAC(f, A)$ using polynomially many membership queries in D and L_ϵ. I will refer to such an algorithm as *efficient*.

Near-optimal evasion is only a *partial* reverse-engineering strategy. Unlike Lowd and Meek's approach, I introduce algorithms that construct queries to provably find an ϵ-$IMAC$

[3] Using the encoded sizes of f, $\mathbf{x}^+$, and $\mathbf{x}^-$ in defining ϵ-$IMAC$ searchable is problematic. For my purposes, it is clear that the encoded size of both $\mathbf{x}^+$ and $\mathbf{x}^-$ is D so it is unnecessary to include additional terms for their size. Further I allow for families of non-parametric classifiers for which the notion of *encoding size* is ill-defined but is also unnecessary for the algorithms I present. In extending beyond linear and parametric family of classifiers, it is not straightforward to define the encoding size of a classifier f. One could use notions such as the *VC-dimension* of $\mathcal{F}$ or its *covering number* but it is unclear why size of the classifier is important in quantifying the complexity of ϵ-$IMAC$ search. Moreover, as I demonstrate in this chapter, there are families of classifiers for which ϵ-$IMAC$ search is polynomial in D and L_ϵ alone.

[4] As is apparent in the algorithms I demonstrate, using $\mathbf{x}^+ = \mathbf{x}^A$ makes the attacker less covert since it is significantly easier to infer the attacker's intentions based on their queries. Covertness is not an explicit goal in ϵ-$IMAC$ search but it would be a requirement of many real-world attackers. However, since the goal of the near-optimal evasion problem is not to design real attacks but rather analyze the best possible attack so as to understand a classifier's vulnerabilities, I exclude any covertness requirement but return to the issue in Section 6.4.2.1.

without *fully* reverse-engineering the classifier; *i.e.*, reconstructing it or estimating the parameters that specify it. Efficient query-based reverse-engineering for $f \in \mathcal{F}$ is sufficient for minimizing A over the estimated negative space. However, generally reverse engineering is an expensive approach for near-optimal evasion, requiring query complexity that is exponential in the feature space dimension D for general convex classes [Rademacher and Goyal, 2009], while finding an ϵ-$IMAC$ need not be—the requirements for finding an ϵ-$IMAC$ differ significantly from the objectives of reverse engineering approaches such as active learning. Both approaches use queries to reduce the size of version space $\hat{\mathcal{F}} \subset \mathcal{F}$; *i.e.*, the set of classifiers consistent with the adversary's membership queries. Reverse engineering approaches minimize the expected number of disagreements between members of $\hat{\mathcal{F}}$. In contrast, to find an ϵ-$IMAC$, the adversary only needs to provide a single instance $\mathbf{x}^\dagger \in \epsilon\text{-}IMAC\,(f, A)$ for all $f \in \hat{\mathcal{F}}$, while leaving the classifier largely unspecified; *i.e.*

$$\bigcap_{f \in \hat{\mathcal{F}}} \epsilon\text{-}IMAC\,(f, A) \neq \emptyset \ .$$

This objective allows the classifier to be unspecified over much of $\mathcal{X}$. I present algorithms for ϵ-$IMAC$ search on a family of classifiers that generally cannot be efficiently reverse engineered—the queries necessarily only elicit an ϵ-$IMAC$; the classifier itself will be underspecified in large regions of $\mathcal{X}$ so these techniques do not reverse engineer the classifier's parameters or decision boundary except in a shrinking region near an ϵ-$IMAC$.

6.1.3 Search Terminology

The notion of near-optimality introduced in Equation (6.3) and the overall near-optimal evasion problem in the previous section is that of *ϵ-multiplicative optimality*; *i.e.*, an ϵ-$IMAC$ must have a cost within a factor of $(1+\epsilon)$ of the MAC. However, the results of this paper can also be immediately adopted for *η-additive optimality* in which the adversary seeks instances with cost no more than $\eta > 0$ *greater* than the MAC. To differentiate between these notions of optimality, I will use the notation ϵ-$IMAC^{(*)}$ to refer to the set in Equation (6.3) and define an analogous set η-$IMAC^{(+)}$ for additive optimality as

$$\eta\text{-}IMAC^{(+)}\,(f, A) \triangleq \left\{ \mathbf{x} \in \mathcal{X}_f^- \;\middle|\; A\,(\mathbf{x}) \leq \eta + \mathrm{MAC}\,(f, A) \right\} \ . \tag{6.4}$$

I use the terms ϵ-$IMAC^{(*)}$ and η-$IMAC^{(+)}$ to refer both to the sets defined in Equation (6.3) and (6.4) as well as the members of them—the usage will be clear from the context.

I consider algorithms that achieve either additive or multiplicative optimality of the family of convex-inducing classifiers. For either notion of optimality one can efficiently use bounds on the MAC to find an ϵ-$IMAC^{(*)}$ or an η-$IMAC^{(+)}$. If there is a negative instance, $\mathbf{x}^-$, with cost C^- and all instances with cost no more than C^+ are positive; *i.e.*, C^- is an upper bound and C^+ is a lower bound on the MAC; *i.e.*, $C^+ \leq MAC\,(f, A) \leq C^-$. The negative instance $\mathbf{x}^-$ is ϵ-multiplicatively optimal if $C_0^-/C_0^+ \leq (1+\epsilon)$ whereas it is η-additively optimal if $C_0^- - C_0^+ \leq \eta$. I consider algorithms that can achieve either additive or multiplicative optimality via binary search. Namely, if the adversary can determine whether an intermediate cost establishes a new upper or lower bound on the MAC, then binary search strategies can iteratively reduce the t^{th} gap between C_t^- and C_t^+ with the fewest steps. I now provide common terminology for the binary search and in Section 6.2 I use convexity to establish a new bound at the t^{th} iteration.

Remark If an algorithm can provide bounds $C^+ \leq MAC\,(f, A) \leq C^-$, then this algorithm has achieved $(C^- - C^+)$-additive optimality and $(\frac{C^-}{C^+} - 1)$-multiplicative optimality.

In the t^{th} iteration of an additive binary search, the *additive gap* between the t^{th} bounds is given by $G_t^{(+)} = C_t^- - C_t^+$ with $G_0^{(+)}$ defined accordingly by the initial bounds C_0^- and C_0^+. The search uses a proposal step of $C_t = \frac{C_t^- + C_t^+}{2}$, a stopping criterion of $G_t^{(+)} \leq \eta$ and achieves η-additive optimality in

$$L_\eta^{(+)} = \left\lceil \log_2 \left(\frac{G_0^{(+)}}{\eta} \right) \right\rceil \tag{6.5}$$

steps. Binary search has the best worst-case query complexity for achieving the η-additive stopping criterion for a unidirectional search (*e.g.*, search along a ray).

Binary search can also be used for multiplicative optimality by searching in exponential space. By rewriting the upper and lower bounds as $C^- = 2^a$ and $C^+ = 2^b$, the multiplicative optimality condition becomes $a - b \leq \log_2(1 + \epsilon)$; *i.e.*, an additive optimality condition. Thus, binary search on the exponent achieves ϵ-multiplicative optimality and does so with the best worst-case query complexity (again in a unidirectional search). The *multiplicative gap* of the t^{th} iteration is $G_t^{(*)} = C_t^- / C_t^+$ with $G_0^{(*)}$ defined accordingly by the initial bounds C_0^- and C_0^+. The t^{th} query is $C_t = \sqrt{C_t^- \cdot C_t^+}$, the stopping criterion is $G_t^{(*)} \leq 1 + \epsilon$ and achieves ϵ-multiplicative optimality in

$$L_\epsilon^{(*)} = \left\lceil \log_2 \left(\frac{\log_2 \left(G_0^{(*)} \right)}{\log_2 (1 + \epsilon)} \right) \right\rceil \tag{6.6}$$

steps. Notice that multiplicative optimality only makes sense when both C_0^- and C_0^+ are strictly positive.

It is also worth noting that both $L_\epsilon^{(+)}$ and $L_\epsilon^{(*)}$ can be instead replaced by $\log\left(\frac{1}{\epsilon}\right)$ for asymptotic analysis. As pointed out by Rubinstein [2010], the near-optimal evasion problem is concerned with the difficulty of making accurate estimates of the MAC, and this difficulty increases as $\epsilon \downarrow 0$. In this sense, clearly $L_\epsilon^{(+)}$ and $\log\left(\frac{1}{\epsilon}\right)$ are asymptotically equivalent. Similarly, comparing $L_\epsilon^{(*)}$ and $\log\left(\frac{1}{\epsilon}\right)$ as $\epsilon \downarrow 0$, the limit of their ratio (by application of L'Hôpital's rule) is

$$\lim_{\epsilon \downarrow 0} \frac{L_\epsilon^{(*)}}{\log\left(\frac{1}{\epsilon}\right)} = 1 \ ;$$

i.e., they are also asymptotically equivalent. Thus, in the following asymptotic results, $L_\epsilon^{(*)}$ can be replaced by $\log\left(\frac{1}{\epsilon}\right)$.

Binary searches for additive and multiplicative optimality differ in their proposal step and their stopping criterion. For additive optimality, the proposal is the arithmetic mean $C_t = \frac{C_t^- + C_t^+}{2}$ and search stops when $G_t^{(+)} \leq \eta$, whereas for multiplicative optimality, the proposal is the geometric mean $C_t = \sqrt{C_t^- \cdot C_t^+}$ and search stops when $G_t^{(*)} \leq 1 + \epsilon$. In the

remainder of this chapter, I will use the fact that binary search is optimal for unidirectional search to search the cost space. At each step in the search, I will use several probes in the instance space $\mathcal{X}$ to determine if the proposed cost is a new upper or lower bound and then continue the binary search accordingly.

6.1.4 Multiplicative vs. Additive Optimality

Additive and multiplicative optimality are intrinsically related by the fact that the optimality condition for multiplicative optimality $C_t^- / C_t^+ \leq 1 + \epsilon$ can be rewritten as additive optimality condition $\log_2\left(C_t^-\right) - \log_2\left(C_t^+\right) \leq \log_2\left(1 + \epsilon\right)$. From this equivalence one can take $\eta = \log_2\left(1 + \epsilon\right)$ and utilize additive optimality criterion on the logarithm of the cost. However, this equivalence also highlights two differences between these notions of optimality.

First, multiplicative optimality only makes sense when both C_0^+ is strictly positive (I use this assumption in my algorithms) whereas additive optimality can still be achieved if $C_0^+ = 0$. In this special case, there is no ϵ-$IMAC^{(*)}$ for any $\epsilon > 0$ unless there is some point $\mathbf{x}^* \in \mathcal{X}_f^-$ that has 0 cost. Practically speaking though, this is a minor hindrance—as I demonstrate in Section 6.2.1.3, there is an algorithm that can efficiently establish any lower bound C_0^+ for any ℓ_p cost if such a lower bound exists.

Second, the additive optimality criterion is not *scale invariant* (*i.e.*, any instance $\mathbf{x}^\dagger$ that satisfies the optimality criterion for cost A also satisfies it for $A'\left(\mathbf{x}\right) = s \cdot A\left(\mathbf{x}\right)$ for any $s > 0$) whereas multiplicative optimality is scale invariant. Additive optimality is, however, *shift invariant* (*i.e.*, any instance $\mathbf{x}^\dagger$ that satisfies the optimality criterion for cost A also satisfies it for $A'\left(\mathbf{x}\right) = s + A\left(\mathbf{x}\right)$ for any $s \geq 0$) whereas multiplicative optimality is not. Scale invariance is typically more salient because if the cost function is also scale invariant (all proper norms are) then the optimality condition is invariant to a rescaling of the underlying feature space; *e.g.*, a change in units for all features. Thus, multiplicative optimality is a unit-less notion of optimality whereas additive optimality is not. The following result is a consequence of additive optimality's lack of scale invariance.

Theorem 6.1. *If for some hypothesis space $\mathcal{F}$, cost function A, and any initial bounds $0 < C_0^+ < C_0^-$ on the $MAC\left(f, A\right)$ for some $f \in \mathcal{F}$, there exists some $\bar{\epsilon} > 0$ such that no efficient query-based algorithm can find an ϵ-$IMAC^{(*)}$ for any $0 < \epsilon \leq \bar{\epsilon}$, then there is no efficient query-based algorithm that can find a η-$IMAC^{(+)}$ for any $0 < \eta \leq \bar{\epsilon} \cdot C_0^-$. As a consequence, if there is $\bar{\epsilon} > 0$ as stated above, then there is generally no efficient query-based algorithm that can find a η-$IMAC^{(+)}$ for any $\eta \geq 0$ since C_0^- could be arbitrarily large.*

Proof. By contraposition. If there is an efficient query-based algorithm that can find a $\mathbf{x} \in \eta$-$IMAC^{(+)}$ for some $0 < \eta \leq \bar{\epsilon} \cdot C_0^-$, then, by definition of η-$IMAC^{(+)}$, $A\left(\mathbf{x}\right) \leq \eta + MAC\left(f, A\right)$. Equivalently, by taking $\eta = \epsilon \cdot MAC\left(f, A\right)$ for some $\epsilon > 0$, this algorithm achieved $A\left(\mathbf{x}\right) \leq (1 + \epsilon) MAC\left(f, A\right)$; *i.e.*, $\mathbf{x} \in \epsilon$-$IMAC^{(*)}$. Moreover, since $MAC\left(f, A\right) \leq C_0^-$, this efficient algorithm is able to find a ϵ-$IMAC^{(*)}$ for some $\epsilon \leq \bar{\epsilon}$. The last remark follows directly from the fact that there is no efficient query-based algorithm for any $0 < \eta \leq \bar{\epsilon} \cdot C_0^-$ and C_0^- could generally be arbitrarily large. □

This theorem demonstrates that additive optimality in near-optimal evasion is an awkward notion. If there is a cost function A for which some family of classifiers $\mathcal{F}$ cannot

be efficiently evaded within any accuracy $0 < \epsilon \leq \bar{\epsilon}$, then the question of whether efficient additive optimality can be achieved for some $\eta > 0$ depends on the scale of the cost function. That is, if η-additive optimality can be efficiently achieved for A, the feature space could rescaled to make η-additive optimality no longer generally efficiently since the rescaling could be chosen to make C_0^- large. This highlights the limitation of the lack of scale invariance in additive optimality: *the units of the cost determine whether a particular level of additive accuracy can be achieved whereas multiplicative optimality is unit-less.* For (weighted) ℓ_1 costs, this is not an issue since, as Section 6.2 shows, there is an efficient algorithm for ϵ-multiplicative optimality for any $\epsilon > 0$. However, as I will demonstrate in Section 6.3, there are ℓ_p costs where this becomes problematic.

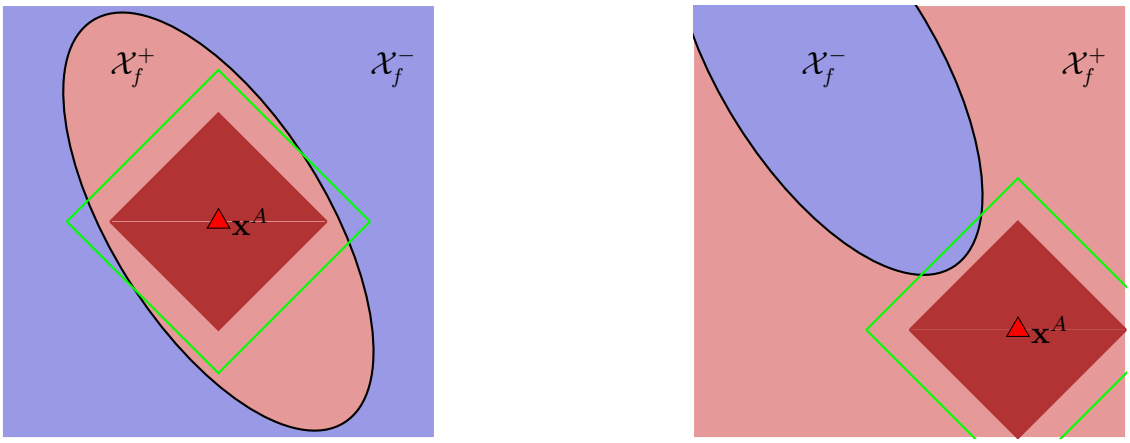
For the remainder of this paper, I primarily only address ϵ-multiplicative optimality for an ϵ-*IMAC* (except where explicitly noted) and define $G_t = G_t^{(*)}$, $C_t = \sqrt{C_t^- \cdot C_t^+}$, and $L_\epsilon = L_\epsilon^{(*)}$. Nonetheless, the algorithms I present can be immediately adapted to additive optimality by simply changing the proposal step, stopping condition, and the definitions of $L_\epsilon^{(*)}$ and G_t, although they may not be generally efficient as discussed above.

6.1.5 The Family of convex-inducing classifiers

Here, I introduce the family of convex-inducing classifiers, $\mathcal{F}^{\text{convex}}$; *i.e.*, the set of classifiers that partition the feature space $\mathcal{X}$ into a positive and negative class, one of which is convex. The convex-inducing classifiers include the linear classifiers studied by Lowd and Meek as well as anomaly detection classifiers using bounded PCA [Lakhina et al., 2004b], anomaly detection algorithms that use hyper-sphere boundaries Bishop [2006], one-class classifiers that predict anomalies by thresholding the log-likelihood of a log-concave (or uni-modal) density function, and quadratic classifiers of the form $\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + c \geq 0$ if $\mathbf{A}$ is semidefinite [see Boyd and Vandenberghe, 2004, Chapter 3]. The convex-inducing classifiers also include complicated bodies such as any intersections of a countable number of halfspaces, cones, or balls.

There is a correspondence between the family of convex-inducing classifiers and the set of all convex sets; *i.e.*, $\mathbb{C} = \{\mathbb{X} \mid \textit{convex}\,(\mathbb{X})\}$. By definition of the convex-inducing classifiers, every classifier $f \in \mathcal{F}^{\text{convex}}$ corresponds to some convex set in $\mathbb{C}$. Further, for any convex set $\mathbb{X} \in \mathbb{C}$, there are at least two trivial classifier that creates that set; namely the classifiers $f_{\mathbb{X}}^{\text{'+'}}(\mathbf{x}) = \mathrm{I}[\mathbf{x} \in \mathbb{X}]$ and $f_{\mathbb{X}}^{\text{'-'}}(\mathbf{x}) = \mathrm{I}[\mathbf{x} \notin \mathbb{X}]$. Thus, in the remainder of this chapter, I will use the existence of particular convex sets to prove results about the convex-inducing classifiers since there is always a corresponding classifier.

It is also worth mentioning the following alternative characterization of the near-optimal evasion problem on the convex-inducing classifiers. For any convex set $\mathbb{C}$ with a non-empty interior let $\mathbf{x}^{(c)}$ be a point in its interior and define the *Minkowski metric* (recentered at $\mathbf{x}^{(c)}$) as $m_{\mathbb{C}}(\mathbf{x}) = \inf\left\{\lambda \;\middle|\; (\mathbf{x} - \mathbf{x}^{(c)}) \in \lambda(\mathbb{C} - \mathbf{x}^{(c)})\right\}$. This function is convex, non-negative, and satisfies $m_{\mathbb{C}}(\mathbf{x}) \leq 1$ if and only if $\mathbf{x} \in \mathbb{C}$. Thus, I can rewrite the definition of the *MAC* of a classifier in terms of the Minkowski metric—if $\mathcal{X}_f^+$ is convex I require $m_{\mathcal{X}_f^+}(\mathbf{x}) > 1$ and if $\mathcal{X}_f^-$ is convex I require $m_{\mathcal{X}_f^-}(\mathbf{x}) \leq 1$. In this way, the near optimal evasion problem (for



(a) (b)

Figure 6.1: Geometry of convex sets and ℓ_1 balls. **(a)** If the positive set $\mathcal{X}_f^+$ is convex, finding an ℓ_1 ball contained within $\mathcal{X}_f^+$establishes a lower bound on the cost, otherwise at least one of the ℓ_1 ball's corners witnesses an upper bound. **(b)** If the negative set $\mathcal{X}_f^-$ is convex, the adversary can establish upper and lower bounds on the cost by determining whether or not an ℓ_1 ball intersects with $\mathcal{X}_f^-$, but this intersection need not include any corner of the ball.

$\mathcal{X}_f^-$ convex) can be rewritten as

$$\begin{aligned} &\text{argmin}_{\mathbf{x}\in\mathcal{X}} \left[A\left(\mathbf{x}\right)\right] \\ &\text{s.t.} \qquad\quad m_{\mathcal{X}_f^-}\left(\mathbf{x}\right) \leq 1 \end{aligned} \tag{6.7}$$

If A is convex, the fact that $m_{\mathbb{C}}\left(\cdot\right)$ is convex makes this a convex program which can be solved by optimizing its Lagrangian

$$\underset{\mathbf{x}\in\mathcal{X},\gamma\in\Re^{0+}}{\text{argmin}} \left[A\left(\mathbf{x}\right) + \gamma\left(1 - m_{\mathcal{X}_f^-}\left(\mathbf{x}\right)\right)\right] \quad .$$

In cases where $m_{\mathcal{X}_f^-}\left(\cdot\right)$ has a closed form, this optimization may have a closed form solution, but generally this approach seems difficult. Instead, I use the special structure of the ℓ_1 cost function to construct efficient search over the family of convex-inducing classifiers.

6.2 Evasion of Convex Classes for ℓ_1 Costs

I generalize ϵ-*IMAC* searchability to the family of convex-inducing classifiers. Restricting $\mathcal{F}$ to be the family of convex-inducing classifiers simplifies ϵ-*IMAC* search. When the negative class $\mathcal{X}_f^-$ is convex, the problem reduces to minimizing a (convex) function A constrained to a convex set—if $\mathcal{X}_f^-$ were known to the adversary, this problem reduces simply to solving a *convex optimization program* [*cf.*, Boyd and Vandenberghe, 2004, Chapter 4]. When the positive class $\mathcal{X}_f^+$ is convex, however, the problem becomes minimizing a (convex) function A outside of a convex set; this is generally a hard problem (*cf.* Section 6.3.1.4 where I show that minimizing ℓ_2 cost can require exponential query complexity). Nonetheless for

certain cost functions A, it is easy to determine whether a particular cost ball $\mathbb{B}^C(A)$ is completely contained within a convex set. This leads to efficient approximation algorithms that I present in this section.

I construct efficient algorithms for query-based optimization of the (weighted) ℓ_1 cost of Equation (6.1) for the convex-inducing classifiers. There is an asymmetry depending on whether the positive or negative class is convex as illustrated in Figure 6.1. When the positive set is convex, determining whether an ℓ_1 ball $\mathbb{B}^C\left(A_1^{(\mathbf{c})}\right) \subset \mathcal{X}_f^+$ only requires querying the vertices of the ball as depicted in Figure 6.1(a). When the negative set is convex, determining whether or not $\mathbb{B}^C\left(A_1^{(\mathbf{c})}\right) \cap \mathcal{X}_f^- = \emptyset$ is non-trivial since the intersection need not occur at a vertex as depicted in Figure 6.1(b). I present an efficient algorithm for optimizing a (weighted) ℓ_1 cost when $\mathcal{X}_f^+$ is convex and a polynomial random algorithm for optimizing any convex cost when $\mathcal{X}_f^-$ is convex.

The algorithms I present achieve multiplicative optimality via binary search. I use Equation (6.6) to define L_ϵ as the number of phases required by binary search[5] to reduce the multiplicative gap to less than $1+\epsilon$. I also use $C_0^- = A_1^{(\mathbf{c})}(\mathbf{x}^-)$ as an initial upper bound on the MAC and assume there is some $C_0^+ > 0$ that lower bounds the MAC (*i.e.*, $\mathbf{x}^A \in int\left(\mathcal{X}_f^+\right)$) . This condition eliminates the case where $\mathbf{x}^A$ is on the boundary of $\mathcal{X}_f^+$ for which $MAC(f, A) = 0$ and ϵ-$IMAC(f, A) = \emptyset$—in this degenerate case, no algorithm can find an ϵ-$IMAC$ since there are negative instances arbitrarily close to $\mathbf{x}^A$.

6.2.1 ϵ-$IMAC$ Search for a Convex $\mathcal{X}_f^+$

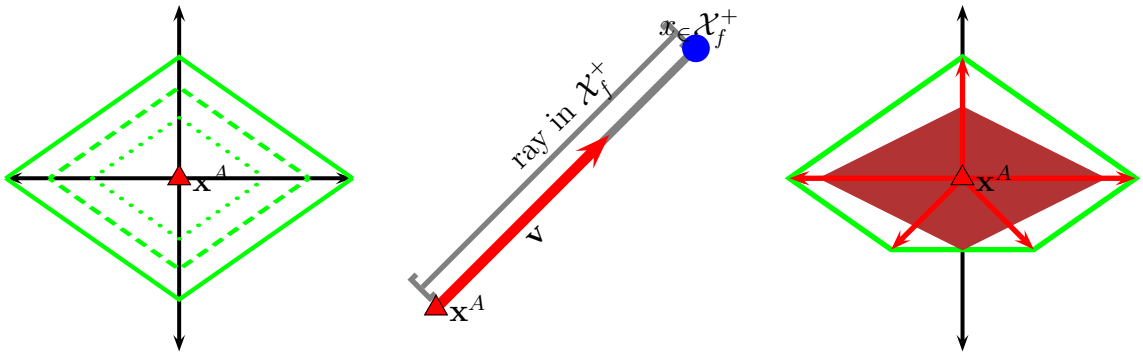
Solving the ϵ-$IMAC$ Search problem when $\mathcal{X}_f^+$ is hard for the general case of optimizing a convex cost A . I demonstrate algorithms for the (weighted) ℓ_1 cost of Equation (6.1) that solve the problem as a binary search. Namely, given initial costs C_0^+ and C_0^- that bound the MAC, I introduce an algorithm that efficiently determines whether $\mathbb{B}^{C_t}(A_1) \subset \mathcal{X}_f^+$ for any intermediate cost $C_t^+ < C_t < C_t^-$. If the ℓ_1 ball is contained in $\mathcal{X}_f^+$, then C_t becomes the new lower bound C_{t+1}^+. Otherwise C_t becomes the new upper bound C_{t+1}^-. Since the objective in Equation (6.3) is to obtain multiplicative optimality, the steps will be $C_t = \sqrt{C_t^+ \cdot C_t^-}$ (for additive optimality, see Section 6.1.3).

The existence of an efficient query algorithm relies on three facts: (1) $\mathbf{x}^A \in \mathcal{X}_f^+$; (2) every weighted ℓ_1 cost C-ball centered at $\mathbf{x}^A$ intersects with $\mathcal{X}_f^-$ only if at least one of its vertices is in $\mathcal{X}_f^-$; and (3) C-balls of weighted ℓ_1 costs only have $2 \cdot D$ vertices. The vertices of the weighted ℓ_1 ball $\mathbb{B}^C(A_1)$ are axis-aligned instances differing from $\mathbf{x}^A$ in exactly one feature (*e.g.*, the d^{th} feature) and can be expressed in the form

$$\mathbf{x}^A \pm \frac{C}{c_d}\mathbf{e}^{(d)} \tag{6.8}$$

which belongs to the C-ball of the weighted ℓ_1 cost (the coefficient $\frac{C}{c_d}$ normalizes for the weight c_d on the d^{th} feature). The second fact is formalized as the following lemma:

[5] As noted in Section 6.1.3, the results of this section can be replicated for additive optimality by using Equation (6.5) for L_ϵ and by using regular binary search proposal and stopping criterion.



(a) (b) (c)

Figure 6.2: The geometry of search. **(a)** Weighted ℓ_1 balls are centered around the target $\mathbf{x}^A$ and have $2 \cdot D$ vertices; **(b)** Search directions in multi-line search radiate from $\mathbf{x}^A$ to probe specific costs; **(c)** In general, the adversary leverages convexity of the cost function when searching to evade. By probing all search directions at a specific cost, the convex hull of the positive queries bounds the ℓ_1 cost ball contained within it.

Lemma 6.2. *For all $C > 0$, if there exists some $\mathbf{x} \in \mathcal{X}_f^-$ that achieves a cost of $C = A_1^{(\mathbf{c})}(\mathbf{x})$, then there is some feature d such that a vertex of the form of Equation* (6.8) *is in $\mathcal{X}_f^-$ (and also achieves cost C by Equation 6.1).*

Proof. Suppose not; then there is some $\mathbf{x} \in \mathcal{X}_f^-$ such that $A_1^{(\mathbf{c})}(\mathbf{x}) = C$ and $\mathbf{x}$ has $M \geq 2$ features that differ from $\mathbf{x}^A$ (if $\mathbf{x}$ differs in one or fewer features it would be of the form of Equation 6.8). Let $\{d_1, \ldots, d_M\}$ be the differing features and let $b_{d_i} = \text{sign}\left(x_{d_i} - x_{d_i}^A\right)$ be the sign of the difference between $\mathbf{x}$ and $\mathbf{x}^A$ along the $d_i{}^{\text{th}}$ feature. For each d_i, let $\mathbf{w}_{d_i} = \mathbf{x}^A + \frac{C}{c_{d_i}} \cdot b_{d_i} \cdot \mathbf{e}^{(d_i)}$ be a vertex of the form of Equation (6.8) which has a cost C (from Equation 6.1). The M vertices $\mathbf{w}_{d_i}$ form an M-dimensional equi-cost simplex of cost C on which $\mathbf{x}$ lies; *i.e.*, $\mathbf{x} = \sum_{i=1}^{M} \alpha_i \mathbf{w}_{d_i}$ for some $0 \leq \alpha_i \leq 1$. If all $\mathbf{w}_{d_i} \in \mathcal{X}_f^+$, then the convexity of $\mathcal{X}_f^+$ implies that all points in their simplex are in $\mathcal{X}_f^+$ and so $\mathbf{x} \in \mathcal{X}_f^+$ which violates the premise. Thus, if any instance in $\mathcal{X}_f^-$ achieves cost C, there is always a vertex of the form Equation (6.8) in $\mathcal{X}_f^-$ that also achieves cost C. □

As a consequence, if all such vertices of any C ball $\mathbb{B}^C(A_1)$ are positive, then all $\mathbf{x}$ with $A_1^{(\mathbf{c})}\mathbf{x} \leq C$ are positive thus establishing C as a lower bound on the MAC. Conversely, if any of the vertices of $\mathbb{B}^C(A_1)$ are negative, then C is an upper bound on MAC. Thus, by simultaneously querying all $2 \cdot D$ equi-cost vertices of $\mathbb{B}^C(A_1)$, the adversary either establishes C as a new lower or upper bound on the MAC. By performing a binary search on C the adversary iteratively halves the multiplicative gap until it is within a factor of $1 + \epsilon$. This yields an ϵ-$IMAC$ of the form of Equation (6.8).

A general form of this multi-line search procedure is presented as Algorithm 6.1 and depicted in Figure 6.2. MultiLineSearch simultaneously searches along all unit-cost search directions in the set $\mathbb{W}$ which contains search directions that radiate from their origin at $\mathbf{x}^A$ and are unit vectors for their cost; *i.e.*, $A(\mathbf{w}) = 1$ for any $\mathbf{w} \in \mathbb{W}$. Of

course, any set of non-normalized search vectors $\{\mathbf{v}\}$ can be transformed into unit search vectors simply by applying a normalization constant of $A(\mathbf{v})^{-1}$ to each. At each step, MultiLineSearch (Algorithm 6.1) issues at most $|\mathbb{W}|$ queries to construct a bounding shell (*i.e.*, the convex hull of these queries will either form an upper or lower bound on the MAC) to determine whether $\mathbb{B}^C(A_1) \subset \mathcal{X}_f^+$. Once a negative instance is found at cost C, the adversary ceases further queries at cost C since a single negative instance is sufficient to establish a lower bound. I call this policy *lazy querying*[6]. Further, when an upper bound is established for a cost C (*i.e.*, a negative vertex is found), the algorithm prunes all directions that were positive at cost C. This pruning is sound; by the convexity assumption these pruned directions are positive for all costs less than the new upper bound C on the MAC so no further queries will be required along such a direction. Finally, by performing a binary search on the cost, MultiLineSearch finds an ϵ-$IMAC$ with no more than $|\mathbb{W}| \cdot L_\epsilon$ queries but at least $|\mathbb{W}| + L_\epsilon$ queries. Thus, this algorithm has a best-case query complexity of $\mathcal{O}(|\mathbb{W}| \cdot L_\epsilon)$ and a worst case query complexity of $\mathcal{O}(|\mathbb{W}| \cdot L_\epsilon)$.

It is worth noting that, in its present form, MultiLineSearch has two implicit assumptions. First, I assume all search directions radiate from a common origin, $\mathbf{x}^A$, and $A(\mathbf{x}^A) = 0$. Without this assumption, the ray-constrained cost function $A(\mathbf{x}^A + s \cdot \mathbf{w})$ is still convex in $s \geq 0$ but not necessarily monotonic as required for binary search. Second, I assume the cost function A is a *positive homogeneous function* along any ray from $\mathbf{x}^A$; *i.e.* $A(\mathbf{x}^A + s \cdot \mathbf{w}) = |s| \cdot A(\mathbf{x}^A + \mathbf{w})$. This assumption allows MultiLineSearch to scale its unit search vectors to achieve the same scaling of their cost. Although the algorithm could be adapted to eliminate these assumptions, the cost functions in Equation (6.1) satisfy both assumptions since they are norms recentered at $\mathbf{x}^A$.

Algorithm 6.2 uses MultiLineSearch for (weighted) ℓ_1 costs by making $\mathbb{W}$ be the vertices of the unit-cost ℓ_1 ball centered at $\mathbf{x}^A$. In this case, the search issues at most $2 \cdot D$ queries to determine whether $\mathbb{B}^C(A_1) \subset \mathcal{X}_f^+$ and thus is $\mathcal{O}(L_\epsilon \cdot D)$. However, MultiLineSearch does not rely on its directions being vertices of the ℓ_1 ball although those vertices are sufficient to span the ℓ_1 ball. Generally, MultiLineSearch is agnostic to the configuration of its search directions and can be adapted for any set of directions that can provide a bound on the cost using the convexity of $\mathcal{X}_f^+$. However, as I show in Section 6.3, the number of search directions required to bound an ℓ_p for $p > 1$ can be exponential in D.

6.2.1.1 K-step Multi-Line Search

Here I present a variant of the multi-line search algorithm that better exploits pruning to reduce the query complexity of Algorithm 6.1. The original MultiLineSearch algorithm is $2 \cdot |\mathbb{W}|$ simultaneous binary searches (*i.e.*, a breadth-first search simultaneously along all search directions). This strategy prunes directions most effectively when the convex body is assymetrically elongated relative to $\mathbf{x}^A$ but fails to prune for symmetrically round bodies. Instead, the algorithm could search sequentially (*i.e.*, a depth-first search of L_ϵ steps along each direction sequentially). This alternative search strategy also obtains a

[6]The search algorithm could continue to query at any distance B^- where there is a known negative instance as it may expedite the pruning of additional search directions early in the search. However, in analyzing the malicious classifier, these additional queries will not lead to further pruning but instead will prevent improvements on the worst-case query complexity as will be demonstrated in Section 6.2.1.1. Thus, the algorithms I present only use lazy querying and only queries at costs below the upper bound C_t^- on the MAC.

Algorithm 6.1. MULTI-LINE SEARCH

$MLS\left(\mathbb{W}, \mathbf{x}^A, \mathbf{x}^-, C_0^+, C_0^-, \epsilon\right)$
$\mathbf{x}^* \leftarrow \mathbf{x}^-$
$t \leftarrow 0$
while $C_t^- / C_t^+ > 1 + \epsilon$ **do begin**
  $C_t \leftarrow \sqrt{C_t^+ \cdot C_t^-}$
  for all $\mathbf{w} \in \mathbb{W}$ **do begin**
    Query: $f_{\mathbf{w}}^t \leftarrow f\left(\mathbf{x}^A + C_t \cdot \mathbf{w}\right)$
    if $f_{\mathbf{w}}^t =$ '$-$' **then begin**
      $\mathbf{x}^* \leftarrow \mathbf{x}^A + C_t \cdot \mathbf{w}$
      Prune $\mathbf{i}$ from $\mathbb{W}$ if $f_{\mathbf{i}}^t =$ '$+$'
      break for-loop
    end if
  end for
  $C_{t+1}^+ \leftarrow C_t^+$ and $C_{t+1}^- \leftarrow C_t^-$
  if $\forall \mathbf{w} \in \mathbb{W}$ $f_{\mathbf{w}}^t =$ '$+$' **then** $C_{t+1}^+ \leftarrow C_t$
  else $C_{t+1}^- \leftarrow C_t$
  $t \leftarrow t + 1$
end while
return: $\mathbf{x}^*$

Algorithm 6.2. CONVEX $\mathcal{X}_f^+$ SEARCH

$ConvexSearch\left(\mathbb{W}, \mathbf{x}^A, \mathbf{x}^-, \epsilon, C^+\right)$
$C^- \leftarrow A\left(\mathbf{x}^-\right)$
$\mathbb{W} \leftarrow \emptyset$
for $i = 1$ to D **do begin**
  $\mathbf{w}^i \leftarrow \frac{1}{c_i} \cdot \mathbf{e}^{(i)}$
  $\mathbb{W} \leftarrow \mathbb{W} \cup \left\{\pm \mathbf{w}^i\right\}$
end for
return: $MLS\left(\mathbb{W}, \mathbf{x}^A, \mathbf{x}^-, C^+, C^-, \epsilon\right)$

Algorithm 6.3. LINEAR $\mathcal{X}_f^+$ SEARCH

$LinearSearch\left(\mathbb{W}, \mathbf{x}^A, \mathbf{x}^-, \epsilon, C^+\right)$
$C^- \leftarrow A\left(\mathbf{x}^-\right)$
$\mathbb{W} \leftarrow \emptyset$
for $i = 1$ to D **do begin**
  $\mathbf{w}^i \leftarrow \frac{1}{c_i} \cdot \mathbf{e}^{(i)}$
  $b_i \leftarrow \text{sign}\left(x_i^- - x_i^A\right)$
  if $b_i = 0$ **then** $\mathbb{W} \leftarrow \mathbb{W} \cup \left\{\pm \mathbf{w}^i\right\}$
  else $\mathbb{W} \leftarrow \mathbb{W} \cup \left\{b_i \mathbf{w}^i\right\}$
end for
return: $MLS\left(\mathbb{W}, \mathbf{x}^A, \mathbf{x}^-, C^+, C^-, \epsilon\right)$

best case of $\mathcal{O}(L_\epsilon + |\mathbb{W}|)$ queries (for a body that is symmetrically round about $\mathbf{x}^A$, it uses L_ϵ queries along the first direction to establish an upper and lower bound within a factor of $1+\epsilon$, then D queries to verify the lower bound) and worst case of $\mathcal{O}(L_\epsilon \cdot |\mathbb{W}|)$ queries (for asymmetrically elongated bodies, in the worst case, the strategy would require L_ϵ queries along each of the D search directions). Surprisingly, these two alternatives have opposite best-case and worst-case convex bodies, which inspired a hybrid approach called K-step MultiLineSearch. This algorithm mixes simultaneous and sequential strategies to achieve a better worst-case query complexity than either pure search strategy[7].

At each phase, the K-step MultiLineSearch (Algorithm 6.4) chooses a single direction $\mathbf{w}$ and queries it for K steps to generate candidate bounds B^- and B^+ on the *MAC*. The algorithm makes substantial progress towards reducing G_t without querying other directions (depth-first). It then iteratively queries all remaining directions at the candidate lower bound B^+ (breadth-first). Again, I use lazy querying and stop as soon as a negative instance is found since B^+ is then no longer a viable lower bound. In this case, although the candidate bound is invalidated, the algorithm can still prune all directions that were positive at B^+ (there will always be at least one such direction). Thus, in every iteration, either the gap is decreased or at least one search direction is pruned. I show that for $K = \lceil\sqrt{L_\epsilon}\rceil$, the algorithm achieves a delicate balance between breadth-first and depth-first approaches to attain a better worst-case complexity than either.

Theorem 6.3. *Algorithm 6.4 will find an ϵ-IMAC with at most $\mathcal{O}\left(L_\epsilon + \sqrt{L_\epsilon}\,|\mathbb{W}|\right)$ queries when $K = \lceil\sqrt{L_\epsilon}\rceil$.*

The proof of this theorem appears in Appendix C.1. As a consequence of Theorem 6.3, finding an ϵ-*IMAC* with Algorithm 6.4 for a (weighted) ℓ_1 cost requires $\mathcal{O}\left(L_\epsilon + \sqrt{L_\epsilon}D\right)$ queries. Further, both Algorithms 6.2 and 6.3 can incorporate K-step MultiLineSearch directly by replacing their function call to MultiLineSearch to K-step MultiLineSearch and using $K = \lceil\sqrt{L_\epsilon}\rceil$.

6.2.1.2 Lower Bound

Here I find lower bounds on the number of queries required by any algorithm to find an ϵ-*IMAC* when $\mathcal{X}_f^+$ is convex for any convex cost function; *e.g.*, Equation (6.1) for $p \geq 1$. Below, I present two theorems, one for both additive and multiplicative optimality. Notably, since an ϵ-*IMAC* uses multiplicative optimality, I incorporate a lower bound $C_0^+ > 0$ on the *MAC* into the theorem statement.

Theorem 6.4. *For any $D > 0$, any positive convex function $A : \Re^D \mapsto \Re^+$, any initial bounds $0 \leq C_0^+ < C_0^-$ on the MAC, and $0 < \eta < C_0^- - C_0^+$, all algorithms must submit at least $\max\{D, L_\eta^{(+)}\}$ membership queries in the worst case to be η-additive optimal on $\mathcal{F}^{\text{convex},\prime+\prime}$.*

Theorem 6.5. *For any $D > 0$, any positive convex function $A : \Re^D \mapsto \Re^+$, any initial bounds $0 < C_0^+ < C_0^-$ on the MAC, and $0 < \epsilon < \frac{C_0^-}{C_0^+} - 1$, all algorithms must submit at least $\max\{D, L_\epsilon^{(*)}\}$ membership queries in the worst case to be ϵ-multiplicatively optimal on $\mathcal{F}^{\text{convex},\prime+\prime}$.*

[7] K-step MultiLineSearch also has a best case of $\mathcal{O}(L_\epsilon + |\mathbb{W}|)$

Algorithm 6.4. K-Step Multi-line Search

$KMLS\left(\mathbb{W}, \mathbf{x}^A, \mathbf{x}^-, C_0^+, C_0^-, \epsilon, K\right)$
$\mathbf{x}^* \leftarrow \mathbf{x}^-$
$t \leftarrow 0$
while $C_t^- / C_t^+ > 1 + \epsilon$ **do begin**
  Choose a direction $\mathbf{w} \in \mathbb{W}$
  $B^+ \leftarrow C_t^+$
  $B^- \leftarrow C_t^-$
  for K steps **do begin**
    $B \leftarrow \sqrt{B^+ \cdot B^-}$
    Query: $f_{\mathbf{w}} \leftarrow f\left(\mathbf{x}^A + B \cdot \mathbf{w}\right)$
    if $f_{\mathbf{w}} =$ '+' **then** $B^+ \leftarrow B$
    else $B^- \leftarrow B$ **and** $\mathbf{x}^* \leftarrow \mathbf{x}^A + B \cdot \mathbf{w}$
  end for
  for all $\mathbf{i} \in \mathbb{W} \setminus \{\mathbf{w}\}$ **do begin**
    Query: $f_{\mathbf{i}}^t \leftarrow f\left(\mathbf{x}^A + (B^+) \cdot \mathbf{i}\right)$
    if $f_{\mathbf{i}}^t =$ '$-$' **then begin**
      $\mathbf{x}^* \leftarrow \mathbf{x}^A + (B^+) \cdot \mathbf{i}$
      Prune $\mathbf{k}$ from $\mathbb{W}$ if $f_{\mathbf{k}}^t =$ '+'
      break for-loop
    end if
  end for
  $C_{t+1}^- \leftarrow B^-$
  if $\forall \mathbf{i} \in \mathbb{W}$ $f_{\mathbf{i}}^t =$ '+' **then** $C_{t+1}^+ \leftarrow B^+$
  else $C_{t+1}^- \leftarrow B^+$
  $t \leftarrow t + 1$
end while
return: $\mathbf{x}^*$

The proof of both of these theorems is in Appendix C.2. Notice, these theorems only apply to $\eta \in \left(0, C_0^- - C_0^+\right)$ and $\epsilon \in \left(0, \frac{C_0^-}{C_0^+} - 1\right)$ respectively. In fact, outside of these intervals the query strategies are trivial. For either $\eta = 0$ or $\epsilon = 0$ no approximation algorithm will terminate. Similarly, for $\eta \geq C_0^- - C_0^+$ or $\epsilon \geq \frac{C_0^-}{C_0^+} - 1$, $\mathbf{x}^-$ is an *IMAC* since it has a cost $A\left(\mathbf{x}^-\right) = C_0^-$, so no queries are required.

Theorems 6.4 and 6.5 show that η-additive and ϵ-multiplicative optimality require $\Omega\left(L_\eta^{(+)} + D\right)$ and $\Omega\left(L_\epsilon^{(*)} + D\right)$ queries respectively. Thus, the K-step MultiLineSearch algorithm (Algorithm 6.4) has close to the optimal query complexity for weighted ℓ_1-costs with its $\mathcal{O}\left(L_\epsilon + \sqrt{L_\epsilon} D\right)$ queries. These bounds also apply to any ℓ_p cost with $p > 1$, but in Section 6.3, I present tighter lower bounds for $p > 1$ that substantially exceed these results for some ranges of ϵ and any range of η.

6.2.1.3 Special Cases

Here I present a number of special cases that require minor modifications to Algorithms 6.1 and 6.4 by adding preprocessing steps.

Revisiting Linear Classifiers: Lowd and Meek originally developed a method for reverse engineering linear classifiers for a (weighted) ℓ_1 cost. First their method isolates a sequence of points from $\mathbf{x}^-$ to $\mathbf{x}^A$ that cross the classifier's boundary and then it estimates the hyperplane's parameters using D local line searches. However, as a consequence of the ability to efficiently minimize our objective when $\mathcal{X}_f^+$ is convex, we immediately have an alternative method for linear classifiers (*i.e.*, half-spaces). In fact, for this special case, as many as half of the search directions can be eliminated using the initial orientation of the hyperplane separating $\mathbf{x}^A$ and $\mathbf{x}^-$. Intuitively, the minimizer in the negative halfspace can only occur along one of the axes of the orthants that contain $\mathbf{x}^-$. This algorithm is presented as Algorithm 6.3. Moreover, because linear classifiers are a special case of convex-inducing classifiers, the K-step MultiLineSearch algorithm improves on the reverse-engineering technique's $\mathcal{O}\left(L_\epsilon \cdot D\right)$ queries and applies to a broader family.

Extending MultiLineSearch Algorithms to $c_d = \infty$ or $c_d = 0$ Weights: In Algorithms 6.2 and 6.3, we reweighted the d^{th} axis-aligned directions by a factor $\frac{1}{c_d}$ to make unit cost vectors by implicitly assuming $c_d \in (0, \infty)$. The case where $c_d = \infty$ (*e.g.* immutable features) is dealt with by simply removing those features from the set of search directions $\mathbb{W}$ used in the MultiLineSearch. In the case when $c_d = 0$ (*e.g.* useless features), MultiLineSearch-like algorithms no longer ensure near-optimality because they implicitly assume that cost balls are bounded sets. If $c_d = 0$, $\mathbb{B}^0\left(A\right)$ is no longer a bounded set and a 0-cost could be achieved if $\mathcal{X}_f^-$ anywhere intersects the subspace spanned by the 0-cost features—this makes near-optimality unachievable unless a negative 0-cost instance can be found. In the worst case, such an instance could be arbitrarily far in any direction within the 0-cost subspace making search for such an instance intractable. Nonetheless, one possible search strategy is to assign all 0-cost features a non-zero weight that decays quickly toward 0 (*e.g.* $c_d = 2^- t$ in the t^{th} iteration) as we repeatedly rerun an MultiLineSearch on the altered objective for T iterations. The algorithm will either find a negative instance that only alters

0-cost features (and hence is a 0-*IMAC*), or it will terminate assuming no such instance exists. This algorithm does not ensure near-optimality but may find a suitable instance with only T runs of a MultiLineSearch.

Lack of an Initial Lower Bound: Thus far, to find a ϵ-*IMAC* the algorithms I presented searched between initial bounds C_0^+ and C_0^-, but, in general, C_0^+ may not be known to a real-world adversary. I now present an algorithm called SpiralSearch that can efficiently establish a lower bound on the *MAC* if one exists. This algorithm performs a halving search on the exponent along a single direction to find a positive example, then queries the remaining directions at that cost. Either the lower bound is verified or directions that were positive can be pruned for the remainder of the search.

Algorithm 6.5. Spiral Search

spiral $\left(\mathbb{W}, \mathbf{x}^A, \mathbf{x}^-, C_0^-, \epsilon\right)$
$t \leftarrow 0$ and $\mathbb{V} \leftarrow \emptyset$
repeat
  Choose a direction $\mathbf{w} \in \mathbb{W}$
  Remove $\mathbf{w}$ from $\mathbb{W}$ and $\mathbb{V} \leftarrow \mathbb{V} \cup \{\mathbf{w}\}$
  Query: $f_{\mathbf{w}} \leftarrow f\left(\mathbf{x}^A + (C_0^-)2^{-2^t}\mathbf{w}\right)$
  if $f_{\mathbf{w}} =$ '$-$' **then begin**
    $\mathbb{W} \leftarrow \mathbb{W} \cup \{\mathbf{w}\}$ and $\mathbb{V} \leftarrow \emptyset$
    $t \leftarrow t + 1$
  end if
until $\mathbb{W} = \emptyset$
$C_0^+ \leftarrow C_0^- \cdot 2^{-2^t}$
return: ($\mathbb{V}$,C_0^+,C_0^-)

At the t^{th} iteration of SpiralSearch a direction is selected and queried at the current lower bound of $(C_0^-)2^{-2^t}$. If the query is positive, that direction is added to the set $\mathbb{V}$ of directions consistent with the lower bound. Otherwise, all directions in $\mathbb{V}$ are discarded and the lower bound is lowered with an exponentially decreasing exponent. Thus, given that some lower bound $C_0^+ > 0$ does exist, one will be found in $\mathcal{O}\left(L_\epsilon + D\right)$ queries and this algorithm can be used as a precursor to any of the previous searches[8] and can be adopted to additive optimality by halving the lower bound instead of the exponent (see Section 6.1.3). Further, the search directions pruned by SprialSearch are also invalid for the subsequent MultiLineSearch so the set $\mathbb{V}$ returned by SprialSearch will be used as the set $\mathbb{W}$ for the subsequent search.

Lack of a Negative Example: The MultiLineSearch algorithms can also naturally be adapted to the case when the adversary has no negative example $\mathbf{x}^-$. This is accomplished by querying ℓ_1 balls of doubly exponentially increasing cost until a negative instance is found. During the t^{th} iteration, the adversary probes along every search direction at a cost $(C_0^+)2^{2^t}$; either all probes are positive (a new lower bound) or at least one is negative (a

[8] If no lower bound on the cost exists, no algorithm can find a ϵ-*IMAC*. As presented, this algorithm would not terminate but in practice, the search would be terminated after sufficiently many iterations.

Algorithm 6.6. INTERSECT SEARCH

IntersectSearch $\left(\mathbb{P}^{(0)}, \mathbb{Q} = \left\{\mathbf{x}^{(j)} \in \mathbb{P}^{(0)}\right\}, C\right)$
for $s = 1$ to T **do begin**
(1) Generate $2N$ samples $\left\{\mathbf{x}^{(j)}\right\}_{j=1}^{2N}$
Choose $\mathbf{x}$ from $\mathbb{Q}$
$\mathbf{x}^{(j)} \leftarrow HitRun\left(\mathbb{P}^{(s-1)}, \mathbb{Q}, \mathbf{x}^{(j)}\right)$
(2) If any $\mathbf{x}^{(j)}$, $A\left(\mathbf{x}^{(j)}\right) \leq C$ terminate the for-loop
(3) Put samples into 2 sets of size N
$\mathbb{R} \leftarrow \left\{\mathbf{x}^{(j)}\right\}_{j=1}^{N}$ and $\mathbb{S} \leftarrow \left\{\mathbf{x}^{(j)}\right\}_{j=N+1}^{2N}$
(4) $\mathbf{z}^{(s)} \leftarrow \frac{1}{N}\sum_{\mathbf{x}^{(j)} \in \mathbb{R}} \mathbf{x}^{(j)}$
(5) Compute $\mathbb{H}^{\left(\mathbf{h}\left(\mathbf{z}^{(s)}\right), \mathbf{z}^{(s)}\right)}$ using Equation (6.10)
(6) $\mathbb{P}^{(s)} \leftarrow \mathbb{P}^{(s-1)} \cap \mathbb{H}^{\left(\mathbf{h}(\mathbf{z}), \mathbf{z}^{(s)}\right)}$
(7) Keep samples in $\mathbb{P}^{(s)}$
$\mathbb{Q} \leftarrow \mathbb{S} \cap \mathbb{P}^{(s)}$
end for
Return: the found $[\mathbf{x}^{(j)}, \mathbb{P}^{(s)}, \mathbb{Q}]$; or No Intersect

Algorithm 6.7. HIT-AND-RUN SAMPLING

$HitRun\left(\mathbb{P}, \left\{\mathbf{y}^{(j)}\right\}, \mathbf{x}^{(0)}\right)$
for $i = 1$ to K **do begin**
(1) Choose a random direction:
$\nu_j \sim \mathrm{N}(0, 1)$
$\mathbf{v} \leftarrow \sum_j \nu_j \cdot \mathbf{y}^{(j)}$
(2) Sample uniformly along $\mathbf{v}$ using rejection sampling:
Choose $\hat{\omega}$ s.t. $\mathbf{x}^{(i-1)} + \hat{\omega} \cdot \mathbf{v} \notin \mathbb{P}$
repeat
$\omega \sim Unif(0, \hat{\omega})$
$\mathbf{x}^{(i)} \leftarrow \mathbf{x}^{(i-1)} + \omega \cdot \mathbf{v}$
$\hat{\omega} \leftarrow \omega$
until $\mathbf{x}^{(i)} \in \mathbb{P}$
end for
Return: $\mathbf{x}^{(K)}$

new upper bound) and search can terminate. Once a negative example is located (having probed for T iterations), we must have $(C_0^+)2^{2^{T-1}} < MAC(f, A) \leq (C_0^+)2^{2^T}$; thus, $T = \left\lceil \log_2 \log_2 \left(\frac{MAC(f,A)}{C_0^+} \right) \right\rceil$. After this preprocessing, the adversary can subsequently perform MULTILINESEARCH with $C_0^+ = 2^{2^{T-1}}$ and $C_0^- = 2^{2^T}$; *i.e.* $\log_2(G_0) = 2^{T-1}$. This precursor step requires at most $|\mathbb{W}| \cdot T$ queries to initialize the MULTILINESEARCH algorithm with a gap such that $L_\epsilon = \left\lceil (T-1) + \log_2 \left(\frac{1}{\log_2(1+\epsilon)} \right) \right\rceil$ according to Equation (6.6).

If there is neither an initial upper bound or lower bound, the adversary can proceed by probing each search direction at unit cost using an additional $|\mathbb{W}|$ queries—this will either establish an upper or lower bound and the adversary can then proceed accordingly.

6.2.2 ϵ-*IMAC* Learning for a Convex $\mathcal{X}_f^-$

In this section, I consider minimizing a convex cost function A (I again focus on weighted ℓ_1 costs in Equation 6.1) when the feasible set $\mathcal{X}_f^-$ is convex. Any convex function can be efficiently minimized within a known convex set (*e.g.*, using an ellipsoid method or interior point method; Boyd and Vandenberghe 2004). However, in the near-optimal evasion problem the convex set is only accessible via membership queries. I use a randomized polynomial algorithm from Bertsimas and Vempala [2004] to minimize the cost function A given an initial point $\mathbf{x}^- \in \mathcal{X}_f^-$. For any fixed cost C^t I use their algorithm to determine (with high probability) whether $\mathcal{X}_f^-$ intersects with $\mathbb{B}^{C^t}(A)$; *i.e.*, whether C^t is a new lower or upper bound on the *MAC*. With high probability, I find an ϵ-*IMAC* in no more than L_ϵ repetitions using binary search. I now focus only on weighted ℓ_1 costs (Equation 6.1) and return to more general cases in Section 6.3.2.

6.2.2.1 Intersection of Convex Sets

I now outline Bertsimas and Vempala's query-based procedure for determining whether two convex sets (*e.g.*, $\mathcal{X}_f^-$ and $\mathbb{B}^{C^t}(A_1)$) intersect. Their IntersectSearch procedure (which I present as Algorithm 6.6) is a randomized ellipsoid method for determining whether there is an intersection between two bounded convex sets: $\mathbb{P}$ is only accessible through membership queries and $\mathbb{B}$ provides a separating hyperplane for any point not in $\mathbb{B}$. They use efficient query-based approaches to uniformly sample from $\mathbb{P}$ to obtain sufficiently many samples such that cutting $\mathbb{P}$ through the centroid of these samples with a separating hyperplane from $\mathbb{B}$ will significantly reduce the volume of $\mathbb{P}$ with high probability. Their technique thus constructs a sequence of progressively smaller feasible sets $\mathbb{P}^{(s)} \subset \mathbb{P}^{(s-1)}$ until either the algorithm finds a point in $\mathbb{P} \cap \mathbb{Q}$ or it is highly likely that the intersection is empty.

As noted earlier, the cost optimization problem reduces to finding the intersection between $\mathcal{X}_f^-$ and $\mathbb{B}^{C^t}(A_1)$. Though $\mathcal{X}_f^-$ may be unbounded, we are minimizing a cost with bounded equi-cost balls, so we can instead use the set $\mathbb{P}^{(0)} = \mathcal{X}_f^- \cap \mathbb{B}^{2R}(A_1; \mathbf{x}^-)$ (where $R = A(\mathbf{x}^-) > C^t$) which is a (convex) subset of $\mathcal{X}_f^-$ that envelops all of $\mathbb{B}^{C^t}(A_1)$ and thus the intersection $\mathcal{X}_f^- \cap \mathbb{B}^{C^t}(A_1)$ if it exists. I also assume that there is some $r > 0$ such that there is an r-ball contained in the convex set $\mathcal{X}_f^-$; *i.e.*, there exists $\mathbf{y} \in \mathcal{X}_f^-$ such that the ball $\mathbb{B}^r(A_1)$ centered at $\mathbf{y}$ is a subset of $\mathcal{X}_f^-$. I now detail this IntersectSearch procedure (Algorithm 6.6).

The foundation of the algorithm is the ability to sample uniformly from an unknown but bounded convex body by means of the hit-and-run random walk technique introduced by Smith [1996] (Algorithm 6.7). Given an instance $\mathbf{x}^{(j)} \in \mathbb{P}^{(s-1)}$, hit-and-run selects a random direction $\mathbf{v}$ through $\mathbf{x}^{(j)}$ (I revisit the selection of $\mathbf{v}$ in Section 6.2.2.2). Since $\mathbb{P}^{(s-1)}$ is a bounded convex set, the set $\mathbb{W} = \left\{\omega \geq 0 \;\middle|\; \mathbf{x}^{(j)} + \omega\mathbf{v} \in \mathbb{P}^{(s-1)}\right\}$ is a bounded interval (*i.e.*, there is some $\hat{\omega} \geq 0$ such that $\mathbb{W} \subset [0, \hat{\omega}]$) which indexes all feasible points along direction $\mathbf{v}$ through $\mathbf{x}^{(j)}$. Sampling ω uniformly from $\mathbb{W}$ yields the next step of the random walk: $\mathbf{x}^{(j)} + \omega\mathbf{v}$. Even though $\hat{\omega}$ is generally unknown, it can be upper bounded and ω can be sampled using rejection sampling along the interval as demonstrated in Algorithm 6.7. Under the appropriate conditions (see Section 6.2.2.2), the hit-and-run random walk generates a sample uniformly from the convex body after $\mathcal{O}^*\left(D^3\right)$ steps[9] [Lovász and Vempala, 2004].

Randomized Ellipsoid Method: I use hit-and-run to obtain $2N$ samples $\left\{\mathbf{x}^{(j)}\right\}$ from $\mathbb{P}^{(s-1)} \subset \mathcal{X}_f^-$ for a single phase of the randomized ellipsoid method. If any satisfy the condition $A\left(\mathbf{x}^{(j)}\right) \leq C^t$, then $\mathbf{x}^{(j)}$ is in the intersection of $\mathcal{X}_f^-$ and $\mathbb{B}^{C^t}(A_1)$ and the procedure is complete. Otherwise, the search algorithm must significantly reduce the size of $\mathbb{P}^{(s-1)}$ without excluding any of $\mathbb{B}^{C^t}(A_1)$ so that sampling concentrates toward the desired intersection (if it exists)—for this we need a separating hyperplane for $\mathbb{B}^{C^t}(A_1)$. For any point $\mathbf{y} \notin \mathbb{B}^{C^t}(A_1)$, the (sub)gradient denoted as $\mathbf{h}(\mathbf{y})$ of the weighted ℓ_1 cost given by

$$\langle \mathbf{h}(\mathbf{y}) \rangle_f = c_f \cdot \operatorname{sign}\left(y_f - x_f^A\right) \quad . \tag{6.9}$$

[9] $\mathcal{O}^*(\cdot)$ denotes the standard complexity notation $\mathcal{O}(\cdot)$ without logarithmic terms.

and thus the hyperplane specified by $\left\{\mathbf{x} \;\middle|\; (\mathbf{x}-\mathbf{y})^\top \mathbf{h}(\mathbf{y})\right\}$ is a separating hyperplane for $\mathbf{y}$ and $\mathbb{B}^{C^t}(A_1)$.

To achieve sufficient progress, the algorithm chooses a point $\mathbf{z} \in \mathbb{P}^{(s-1)}$ so that cutting $\mathbb{P}^{(s-1)}$ through $\mathbf{z}$ with the hyperplane $\mathbf{h}(\mathbf{z})$ eliminates a significant fraction of $\mathbb{P}^{(s-1)}$. To do so, $\mathbf{z}$ must be centrally located within $\mathbb{P}^{(s-1)}$. We use the empirical centroid of the half of the samples in $\mathbb{R}$: $\mathbf{z} = \frac{1}{N}\sum_{\mathbf{x}\in\mathbb{R}} \mathbf{x}$ (the other half will be used in Section 6.2.2.2). We cut $\mathbb{P}^{(s-1)}$ with the hyperplane $\mathbf{h}(\mathbf{z})$ through $\mathbf{z}$; *i.e.*, $\mathbb{P}^{(s)} = \mathbb{P}^{(s-1)} \cap \mathbb{H}^{(\mathbf{h}(\mathbf{z}),\mathbf{z})}$ where $\mathbb{H}^{(\mathbf{h}(\mathbf{z}),\mathbf{z})}$ is the halfspace

$$\mathbb{H}^{(\mathbf{h}(\mathbf{z}),\mathbf{z})} = \left\{\mathbf{x} \;\middle|\; \mathbf{x}^\top \mathbf{h}(\mathbf{z}) < \mathbf{z}^\top \mathbf{h}(\mathbf{z})\right\} \quad . \tag{6.10}$$

As shown by Bertsimas and Vempala, this cut achieves $vol\left(\mathbb{P}^{(s)}\right) \leq \frac{2}{3} vol\left(\mathbb{P}^{(s-1)}\right)$ with high probability if $N = \mathcal{O}^*(D)$ and $\mathbb{P}^{(s-1)}$ is *near-isotropic* (see Section 6.2.2.2). Since the ratio of volumes between the initial circumscribing and inscribed balls of the feasible set is $\left(\frac{R}{r}\right)^D$, the algorithm can terminate after $T = \mathcal{O}\left(D \log\left(\frac{R}{r}\right)\right)$ unsuccessful iterations with a high probability that the intersection is empty.

Because every iteration in Algorithm 6.6 requires $N = \mathcal{O}^*(D)$ samples, each of which need $K = \mathcal{O}^*\left(D^3\right)$ random walk steps, and there are $\mathcal{O}^*(D)$ iterations, the total number of membership queries required by Algorithm 6.6 is $\mathcal{O}^*\left(D^5\right)$.

6.2.2.2 Sampling from a Queryable Convex Body

In the randomized ellipsoid method, random samples are used for two purposes: estimating the convex body's centroid and maintaining the conditions required for the HIT-AND-RUN sampler to efficiently generate points uniformly from a sequence of shrinking convex bodies. Until now, I assumed the HIT-AND-RUN random walk efficiently produces uniformly random samples from any bounded convex body $\mathbb{P}$ using $K = \mathcal{O}^*\left(D^3\right)$ membership queries. However, if the body is asymmetrically elongated, randomly selected directions will rarely align with the long axis of the body and the random walk will take small steps (relative to the long axis) and mix slowly in $\mathbb{P}$. For the sampler to mix effectively, the convex body $\mathbb{P}$ has to be sufficiently round, or more formally near-isotropic; *i.e.*, for any unit vector $\mathbf{v}$,

$$\frac{1}{2} vol(\mathbb{P}) \;\leq\; \mathrm{E}_{\mathbf{x}\sim\mathbb{P}}\left[\left(\mathbf{v}^\top (\mathbf{x} - \mathrm{E}_{\mathbf{x}\sim\mathbb{P}}[\mathbf{x}])\right)^2\right] \;\leq\; \frac{3}{2} vol(\mathbb{P}) \quad . \tag{6.11}$$

If the body is not near-isotropic, $\mathcal{X}$ can be rescaled with an appropriate affine transformation $\mathbf{T}$ so the resulting body $\mathbb{P}' = \{\mathbf{T}\mathbf{x} \mid \mathbf{x} \in \mathbb{P}\}$ is near-isotropic. With sufficiently many samples from $\mathbb{P}$ we can estimate $\mathbf{T}$ as their empirical covariance matrix. Instead, we rescale $\mathcal{X}$ implicitly using a technique described by Bertsimas and Vempala [2004]. We maintain a set $\mathbb{Q}$ of sufficiently many uniform samples from the body $\mathbb{P}^{(s)}$ and in the HIT-AND-RUN algorithm (Algorithm 6.7) we sample the direction $\mathbf{v}$ based on this set. Intuitively, because the samples in $\mathbb{Q}$ are distributed uniformly in $\mathbb{P}^{(s)}$, the directions we sample based on the points in $\mathbb{Q}$ implicitly reflect the covariance structure of $\mathbb{P}^{(s)}$. This is equivalent to sampling the direction $\mathbf{v}$ from a normal distribution with zero mean and covariance of $\mathbb{P}$.

Further, the set $\mathbb{Q}$ must retain sufficiently many samples from $\mathbb{P}^{(s)}$ after each cut: $\mathbb{P}^{(s)} \leftarrow \mathbb{P}^{(s-1)} \cap \mathbb{H}^{\left(\mathbf{h}\left(\mathbf{z}^{(s)}\right),\mathbf{z}^{(s)}\right)}$. To do so, we initially resample $2N$ points from $\mathbb{P}^{(s-1)}$ using HIT-AND-RUN—half of these, $\mathbb{R}$, are used to estimate the centroid $\mathbf{z}^{(s)}$ for the cut and the other half,

$\mathbb{S}$, are used to repopulate $\mathbb{Q}$ after the cut. Because $\mathbb{S}$ contains independent uniform samples from $\mathbb{P}^{(s-1)}$, those in $\mathbb{P}^{(s)}$ after the cut constitute independent uniform samples from $\mathbb{P}^{(s)}$ (*i.e.*, rejection sampling). By choosing N sufficiently large, the cut will be sufficiently deep and there will be sufficiently many points to resample $\mathbb{P}^{(s)}$ after the cut.

Finally, the algorithm also re-queries an initial set $\mathbb{Q}$ of uniform samples from $\mathbb{P}^{(0)}$ but, in the near-optimal evasion problem, only a single point $\mathbf{x}^- \in \mathcal{X}_f^-$ is known. Fortunately, there is an iterative procedure for putting the initial convex set $\mathbb{P}^{(0)}$ into a near-isotropic position from which we obtain $\mathbb{Q}$. The RoundingBody algorithm described in Lovász and Vempala [2003] uses $\mathcal{O}^*\left(D^4\right)$ membership queries to transforms the convex body into a near-isotropic position. We use this as a preprocessing step for Algorithms 6.6 and 6.8; that is, given $\mathcal{X}_f^-$ and $\mathbf{x}^- \in \mathcal{X}_f^-$ we make $\mathbb{P}^{(0)} = \mathcal{X}_f^- \cap \mathbb{B}^{2R}\left(A_1; \mathbf{x}^-\right)$ and then use the RoundingBody algorithm to produce an initial uniform sample $\mathbb{Q} = \left\{\mathbf{x}^{(j)} \in \mathbb{P}^{(0)}\right\}$. These sets are then the inputs to the search algorithms.

6.2.2.3 Optimization over ℓ_1 Balls

I now revisit the outermost optimization loop (for searching the minimum feasible cost) of the algorithm to optimize the naive approach which repeats the intersection search at each step of the binary search over cost balls. First, since $\mathbf{x}^A$, $\mathbf{x}^-$ and $\mathbb{Q}$ are the same for every iteration of the optimization procedure, we only need to run the RoundingBody procedure once as a preprocessing step rather than running it as a preprocessing step every time IntersectSearch is invoked. The set of samples $\mathbb{Q} = \left\{\mathbf{x}^{(j)} \in \mathbb{P}^{(0)}\right\}$ produced by RoundingBody are sufficient to initialize the IntersectSearch at each stage of the binary search over C^t. Second, the separating hyperplane $\mathbf{h}\left(\mathbf{y}\right)$ given by Equation (6.9) does not depend on the target cost C^t but only on $\mathbf{x}^A$, the common center of all the ℓ_1 balls. In fact, this separating hyperplane through the point $\mathbf{y}$ is valid for all weighted ℓ_1-balls of cost $C < A\left(\mathbf{y}\right)$. Further, if $C < C^t$, then $\mathbb{B}^C\left(A_1\right) \subset \mathbb{B}^{C^t}\left(A_1\right)$. Thus, the final state from a successful call to IntersectSearch for the C^t-ball can be used as the starting state for any subsequent call to IntersectSearch for all $C < C^t$. These improvements are reflected in the final procedure SetSearch in Algorithm 6.8 (as with previous binary search procedures, this algorithm can be trivially adapted for η-additive optimality simply by changing it's stopping criterion and proposal step as explained in Section 6.1.3)—the total number of queries required is also $\mathcal{O}^*\left(D^5\right)$ since the algorithm only takes L_ϵ binary search steps.

6.3 Evasion for General ℓ_p Costs

Here I further extend ϵ-*IMAC* searchability over the family of convex-inducing classifiers to the full family of ℓ_p costs for any $0 < p \leq \infty$. As I demonstrate in this section, many ℓ_p costs are not generally ϵ-*IMAC* searchable for all $\epsilon > 0$ over the family of convex-inducing classifiers (*i.e.*, I show that finding an ϵ-*IMAC* for this family can require exponentially many queries in D and L_ϵ). In fact, only the weighted ℓ_1 costs have known (randomized) polynomial query strategies when either the positive or negative set is convex.

Algorithm 6.8. CONVEX $\mathcal{X}_f^-$ SET SEARCH

$SetSearch\left(\mathbb{P}, \mathbb{Q} = \left\{\mathbf{x}^{(j)} \in \mathbb{P}\right\}, C_0^-, C_0^+, \epsilon\right)$
$\mathbf{x}^* \leftarrow \mathbf{x}^-$ and $t \leftarrow 0$
while $C_t^- / C_t^+ > 1 + \epsilon$ **do begin**
  $C_t \leftarrow \sqrt{C_t^- \cdot C_t^+}$
  $[\mathbf{x}^*, \mathbb{P}', \mathbb{Q}'] \leftarrow IntersectSearch\left(\mathbb{P}, \mathbb{Q}, C\right)$
  if intersection found **then begin**
    $C_{t+1}^- \leftarrow A\left(\mathbf{x}^*\right)$ and $C_{t+1}^+ \leftarrow C_t^+$
    $\mathbb{P} \leftarrow \mathbb{P}'$ and $\mathbb{Q} \leftarrow \mathbb{Q}'$
  else
    $C_{t+1}^- \leftarrow C_t^-$ and $C_{t+1}^+ \leftarrow C_t$
  end if
  $t \leftarrow t + 1$
end while
Return: $\mathbf{x}^*$

6.3.1 Convex Positive Set

Here, I explore the ability of the MULTILINESEARCH and K-STEP MULTILINESEARCH algorithms presented in Section 6.2.1 to find solutions to the near-optimal evasion problem for ℓ_p cost functions with $p \neq 1$. Particularly for $p > 1$ I explore the consequences of using the MULTILINESEARCH algorithms using more search directions than just the $2 \cdot D$ axis-aligned directions. Figure 6.3 demonstrates how queries can be used to construct upper and lower bounds on general ℓ_p costs. The following lemma also summarizes well-known bounds on general ℓ_p costs using an ℓ_1 cost.

Lemma 6.6. *The largest ℓ_p ($p > 1$) ball enclosed within a C-cost ℓ_1 ball has a cost of $C \cdot D^{\frac{1-p}{p}}$ and for $p = \infty$ the cost is $C \cdot D^{-1}$.*

Proof. By symmetry, the point $\mathbf{x}^*$ on the simplex $\left\{\mathbf{x} \in \Re^D \;\middle|\; \sum_{i=1}^D x_i = 1, x_i \geq 0 \forall i\right\}$ that minimizes the ℓ_p norm for any $p > 1$ is

$$\mathbf{x}^* = \frac{1}{D}\left(1, 1, \ldots, 1\right) \quad .$$

The ℓ_p norm (cost) of the minimizer is

$$\begin{aligned} \left\|\mathbf{x}^*\right\|_p &= \frac{1}{D}\left(\sum_{i=1}^D 1^p\right)^{1/p} \\ &= \frac{1}{D} D^{1/p} \\ &= D^{\frac{1-p}{p}} \end{aligned}$$

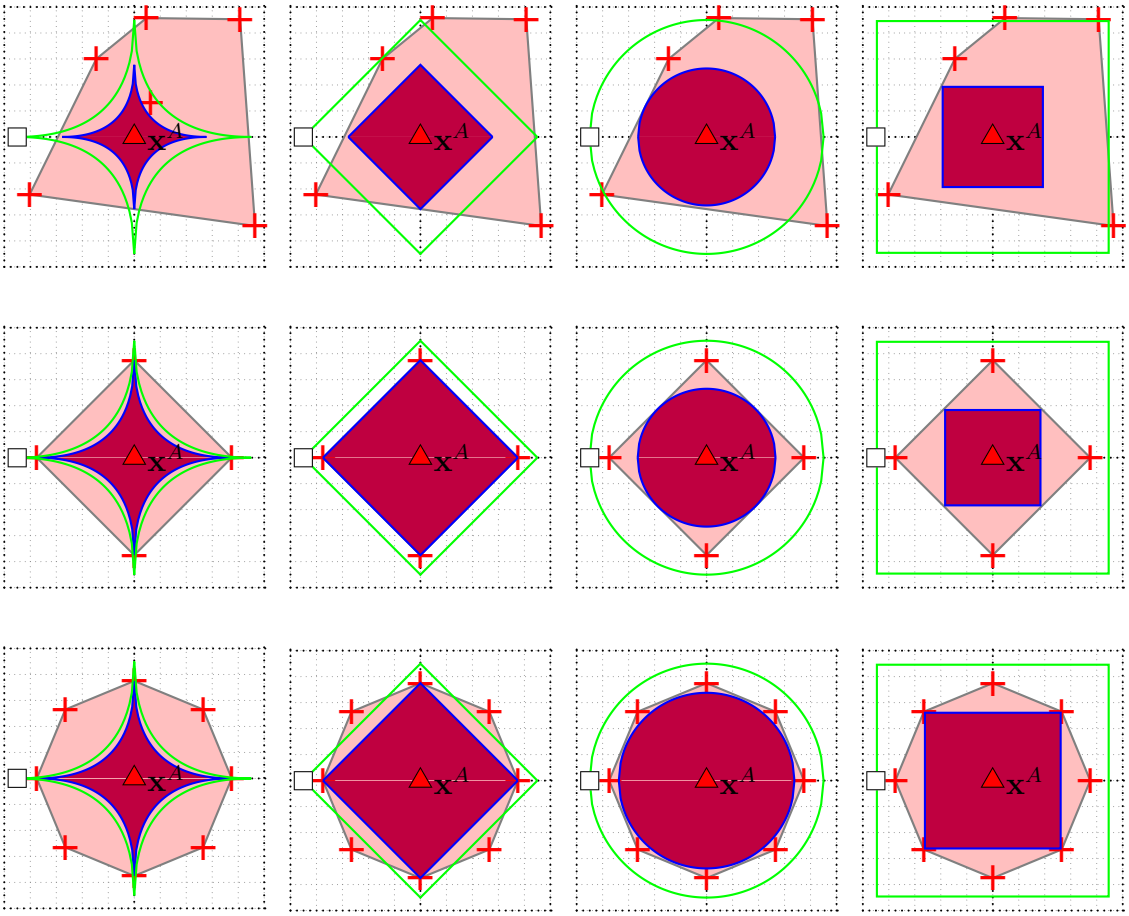


Figure 6.3: Convex hull for a set of queries and the resulting bounding balls for several ℓ_p costs. Each row represents a unique set of positive (red '+' points) and negative (green '−' points) queries and each column shows the implied upper bound (in green) and lower bound (in blue) for a different ℓ_p cost. In the first row, the body is defined by a random set of 7 queries, in the second, the queries are along the coordinate axes, and in the third, the queries are around a circle.

for $p \in (1, \infty)$ and is otherwise

$$\begin{aligned} \|\mathbf{x}^*\|_\infty &= \max\left[\frac{1}{D}, \frac{1}{D}, \ldots, \frac{1}{D}\right] \\ &= D^{-1} \ . \end{aligned}$$

□

6.3.1.1 Bounding ℓ_p Balls

In general, suppose one probes along some set of M unit directions and eventually there is at least one negative point supporting an upper bound of C_0^- and M positive points supporting at a cost of C_0^+. However, the lower bound provided by those M positive points

is the cost of the largest ℓ_p cost ball that fits entirely within their convex hull; let's say this cost is $C^\dagger \leq C_0^+$. To achieve ϵ-multiplicative optimality, we need

$$\frac{C_0^-}{C^\dagger} \leq 1 + \epsilon \ ,$$

which can be rewritten as

$$\left(\frac{C_0^-}{C_0^+}\right)\left(\frac{C_0^+}{C^\dagger}\right) \leq 1 + \epsilon \ .$$

This divides the problem into two parts. The first ratio C_0^-/C_0^+ is controlled solely by the accuracy ϵ achieved by running the MultiLineSearch algorithm for L_ϵ steps whereas the second ratio $C_0^+/C^\dagger$ depends only on how well the ℓ_p ball is approximated by the convex hull of the M search directions. These two ratios separate the search task into choosing M and L_ϵ sufficiently so that their product is less than $1+\epsilon$. First we select parameters $\alpha \geq 0$ and $\beta \geq 0$ such that $(1+\alpha)(1+\beta) \leq 1+\epsilon$. Then we choose M so that

$$\frac{C_0^+}{C^\dagger} = 1 + \beta$$

and use L_α steps so that MultiLineSearch with M directions will achieve

$$\frac{C_0^-}{C_0^+} = 1 + \alpha \ .$$

This process describes a generalized MultiLineSearch that achieves ϵ-multiplicative optimality for costs whose cost-balls are not spanned by the hull of equi-cost probes along the M search directions.

In the case of $p = 1$, I demonstrated in Section 6.2.1 that choosing the $M = 2 \cdot D$ axis-aligned directions $\left\{\pm \mathbf{e}^{(d)}\right\}$ spans the ℓ_1 ball so that $C_0^+/C^\dagger = 1$ (*i.e.*, $\beta = 0$). Thus, choosing $\alpha = \epsilon$, recovers the original multi-line search result.

I now address costs where $\beta > 0$. For a MultiLineSearch algorithm to be efficient, it is necessary that $\frac{C_0^+}{C^\dagger} = 1 + \beta$ can be achieved with polynomially-many search directions (in D and L_ϵ) for some $\beta \leq \epsilon$; otherwise, $(1+\alpha)(1+\beta) > 1+\epsilon$ and the MultiLineSearch approach cannot succeed. Thus, I quantify how many search directions (or queries) are required to achieve

$$\frac{C_0^+}{C^\dagger} \leq 1 + \epsilon \ .$$

Note that this ratio is independent of the relative size of these costs, so without loss of generality I will only consider bounds for unit-cost balls. Thus, I compute the largest value of $C^\dagger$ that can be achieved for the unit-cost ℓ_p ball (*i.e.*, let $C_0^+ = 1$) within the convex hull of M queries. In particular, I will quantify how many queries are required to achieve

$$C^\dagger \geq \frac{1}{1+\epsilon} \ . \tag{6.12}$$

If this can be achieved with only polynomially-many queries, then the generalized MultiLineSearch approach is efficient. More generally,

Lemma 6.7. *If there exists a configuration of M unit search directions with a convex hull that yields a bound $C^\dagger$ for the cost function A, then* MultiLineSearch *algorithms can use those search directions to achieve ϵ-multiplicative optimality with a query complexity that is polynomial in M and $L_\epsilon^{(*)}$ for any*

$$\epsilon > \frac{1}{C^\dagger} - 1 \ .$$

Moreover, if the M search directions yield $C^\dagger = 1$ for the cost function A, then MultiLineSearch *algorithms can achieve ϵ-multiplicative optimality with a query complexity that is polynomial in M and $L_\epsilon^{(*)}$ for any $\epsilon > 0$.*

Notice that this lemma also reaffirms that for $p = 1$ using the $M = 2 \cdot D$ axis-aligned directions allows MultiLineSearch algorithms to achieve ϵ-multiplicative optimality for any $\epsilon > 0$ with a query complexity that is polynomial in M and $L_\epsilon^{(*)}$ since in this case $C^\dagger = 1$. Also recall that as a consequence of Theorem 6.1, if a particular multiplicative accuracy ϵ cannot be efficiently achieved, then additive optimality cannot be generally achieved for any additive accuracy $\eta > 0$.

6.3.1.2 Multi-line Search for $0 < p < 1$

A simple result holds here. Namely, since the unit ℓ_1 ball bounds any unit ℓ_p balls with $0 < p < 1$ one can achieve $C_0^+/C^\dagger = 1$ using only the $2 \cdot D$ axis-aligned search directions. Thus, evasion is efficient for every $0 < p < 1$ for any value of $\epsilon > 0$. Whether or not any ℓ_p $(0 < p < 1)$ cost function can be efficiently searched with fewer search directions is an open question.

6.3.1.3 Multi-line Search for $p > 1$

For this case, one can trivially use the ℓ_1 bound on ℓ_p balls as summarized by the following corollary.

Corollary 6.8. *For $1 < p < \infty$ and $\epsilon \in \left(D^{\frac{p-1}{p}} - 1, \infty\right)$ any multi-line search algorithm can achieve ϵ-multiplicative optimality on A_p using $M = 2 \cdot D$ search directions. Similarly for $\epsilon \in (D - 1, \infty)$ any multi-line search algorithm can achieve ϵ-multiplicative optimality on A_∞.*

Proof. From Lemma 6.6, the largest co-centered ℓ_p ball contained within the unit ℓ_1 ball has radius $D^{\frac{1-p}{p}}$ cost (or D for $p = \infty$). The bounds on ϵ then follow from Lemma 6.7. □

Unfortunately, this result only applies for a range of ϵ that grows with D, which is insufficient for ϵ-*IMAC* searchability. In fact, for some fixed values of ϵ, there is no query-based strategy that can bound ℓ_p costs using polynomially-many queries in D as the following result shows.

Theorem 6.9. *For $p > 1$, $D > 0$, any initial bounds $0 < C_0^+ < C_0^-$ on the MAC, and $\epsilon \in \left(0, 2^{\frac{p-1}{p}} - 1\right)$ (or $\epsilon \in (0, 1)$ for $p = \infty$), all algorithms must submit at least $\alpha_{p,\epsilon}^D$ membership queries (for some constant $\alpha_{p,\epsilon} > 1$) in the worst case to be ϵ-multiplicatively optimal on $\mathcal{F}^{\text{convex},'+'}$ for ℓ_p costs.*

The proof of this theorem is provided in Appendix C.3 and the definitions of $\alpha_{p,\epsilon}$ and $\alpha_{\infty,\epsilon}$ are provided by Equations (C.7) and (C.8), respectively. A consequence of this result is that there is no query-based algorithm that can efficiently find an ϵ-*IMAC* of any ℓ_p cost ($p > 1$) for any $0 < \epsilon < 2^{\frac{p-1}{p}}$ (or $0 < \epsilon < 1$ for $p = \infty$) on the family $\mathcal{F}^{\text{convex,'+'}}$. However, from Theorem 6.8 and Lemma 6.7, multi-line search type algorithms efficiently find the ϵ-*IMAC* of any ℓ_p cost ($p > 1$) for any $\epsilon \in \left(D^{\frac{p-1}{p}} - 1, \infty\right)$ (or $D - 1 < \epsilon < \infty$ for $p = \infty$). It is generally unclear if efficient algorithms exist for any values of ϵ between these intervals, but in the following section, I derive a stronger bound for the case $p = 2$.

6.3.1.4 Multi-line Search for $p = 2$

Theorem 6.10. *For any $D > 1$, any initial bounds $0 < C_0^+ < C_0^-$ on the MAC, and $0 < \epsilon < \frac{C_0^-}{C_0^+} - 1$, all algorithms must submit at least $\alpha_\epsilon^{\frac{D-2}{2}}$ membership queries (where $\alpha_\epsilon = \frac{(1+\epsilon)^2}{(1+\epsilon)^2-1} > 1$) in the worst case to be ϵ-multiplicatively optimal on $\mathcal{F}^{\text{convex,'+'}}$ for ℓ_2 costs.*

The proof of this result is in Appendix C.4.

This result says that there is no algorithm can generally achieve ϵ-multiplicative optimality for ℓ_2 costs for any fixed $\epsilon > 0$ using only polynomially-many queries in D since the ratio $\frac{C_0^-}{C_0^+}$ could be arbitrarily large. It may appear that Theorem 6.10 contradicts Corollary 6.8. However, Corollary 6.8 only applies for a range of ϵ that depends on D; *i.e.*, $\epsilon > \sqrt{D} - 1$. Interestingly, by substituting this lower bound on ϵ into the bound given by Theorem 6.10, the number of required queries for $\epsilon > \sqrt{D} - 1$ need only be

$$M \quad \geq \quad \left(\frac{(1+\epsilon)^2}{(1+\epsilon)^2 - 1}\right)^{\frac{D-2}{2}} \quad = \quad \left(\frac{D}{D-1}\right)^{\frac{D-2}{2}} \quad ,$$

which is a monotonically increasing function in D that asymptotes at $\sqrt{e} \approx 1.64$. Thus, Theorem 6.10 and Corollary 6.8 are in agreement since for $\epsilon > \sqrt{D} - 1$, the former only requires at least 2 queries, which is a trivial bound for all D.

A Tighter Bound: The bound derived for Lemma A.1 was sufficient to demonstrate that there is no algorithm can generally achieve ϵ-multiplicative optimality for ℓ_2 costs for any fixed $\epsilon > 0$. It is, however, possible to construct a tighter lower bound on the number of queries required for ℓ_2 costs although it is not easy to express this result as an exponential in D. A straightforward way to construct a better lower bound is to make a tighter upper bound on the integral $\int_0^\phi \sin^D(t)\,\mathrm{d}t$ as is suggested in Appendix A.1. Namely, the result given in Equation (A.3) upper bounds this integral by

$$\frac{\sin^{D+1}(\phi)}{(D+1)\cos(\phi)} \quad ,$$

which is tighter for large D and $\phi < \frac{\pi}{2}$. Applying this bound to the covering number result of Theorem 6.10 achieves the following bound on the number of queries required to achieve

multiplicative optimality.

$$M \geq \frac{\sqrt{\pi}}{1+\epsilon} \cdot \frac{D \cdot \Gamma\left(\frac{D+1}{2}\right)}{\Gamma\left(1+\frac{D}{2}\right)} \left(\frac{(1+\epsilon)^2}{(1+\epsilon)^2 - 1} \right)^{\frac{D-1}{2}} \quad . \tag{6.13}$$

While not as obvious as the result presented in Appendix C.4, this bound is also exponential in D for any ϵ. Also, as with the previous result, this bound does not contradict the polynomial result for $\epsilon \geq \sqrt{D} - 1$. For $D = 1$ Equation 6.13 requires exactly 2 queries (in exact agreement with the number of queries required to bound an ℓ_2 ball in 1-dimension), for $D = 2$ it requires more than π queries (whereas at least 4 queries are actually required) and for $D > 2$ the bound asymptotes at $\sqrt{2e\pi} \approx 4.13$ queries. Again, this tighter bound does not contradict the efficient result achieved by bounding ℓ_2 balls with ℓ_1 balls.

6.3.2 Convex Negative Set

Algorithm 6.8 generalizes immediately to all weighted ℓ_p costs ($p \geq 1$) centered at $\mathbf{x}^A$ since these costs are convex. For these costs an equivalent separating hyperplane for $\mathbf{y}$ can be used in place of Equation (6.9). They are given by the equivalent (sub)-gradients for ℓ_p cost balls:

$$\begin{aligned} h_{p,d}^{(\mathbf{y})} &= c_d \operatorname{sign}\left(y_d - x_d^A\right) \cdot \left(\frac{|y_d - x_d^A|}{A_p^{(\mathbf{c})}(\mathbf{y})} \right)^{p-1} \quad , \\ h_{\infty,d}^{(\mathbf{y})} &= c_d \operatorname{sign}\left(y_d - x_d^A\right) \cdot \mathrm{I}\left[|y_d - x_d^A| = A_p^{(\mathbf{c})}(\mathbf{y}) \right] \quad . \end{aligned}$$

By only changing the cost function A and the separating hyperplane $\mathbf{h}(\mathbf{y})$ used for the halfspace cut in Algorithms 6.6 and 6.8, the randomized ellipsoid method can be applied for any weighted ℓ_p cost $A_p^{(\mathbf{c})}$.

For more general convex costs A, every C-cost ball is a convex set (*i.e.*, the sublevel set of a convex function is a convex set; Boyd and Vandenberghe see 2004, Chapter 3) and thus has a separating hyperplane. Further, since for any $D > C$, $\mathbb{B}^C(A) \subset \mathbb{B}^D(A)$, the separating hyperplane of the D-cost ball is also a separating hyperplane of the C cost ball and can be re-used in Algorithm 6.8. Thus, this procedure is applicable for any convex cost function A so long as one can compute the separating hyperplanes of any cost ball of A for any point $\mathbf{y}$ not in the cost ball.

For non-convex costs A such as weighted ℓ_p costs with $0 < p < 1$, minimization over a convex set $\mathcal{X}_f^-$ is generally hard. However, there may be special cases when minimizing such a cost can be accomplished efficiently.

6.4 Summary and Future Work

Here, I primarily studied membership query algorithms that efficiently accomplish ϵ-*IMAC* search for convex-inducing classifiers with weighted ℓ_1 costs. When the positive class is convex, I demonstrate efficient techniques that outperform the previous reverse-engineering approaches for linear classifiers. When the negative class is convex, I apply the randomized ellipsoid method introduced by Bertsimas and Vempala to achieve efficient ϵ-*IMAC* search.

If the adversary is unaware of which set is convex, he can trivially run both searches to discover an ϵ-$IMAC$ with a combined polynomial query complexity; thus, for ℓ_1 costs, the family of convex-inducing classifiers can be efficiently evaded by an adversary; *i.e.*, this family is ϵ-$IMAC$ searchable.

Further, I also extended the study of convex-inducing classifiers to general ℓ_p costs. I showed that $\mathcal{F}^{convex}$ is only ϵ-$IMAC$ searchable for both positive and negative convexity for any $\epsilon > 0$ if $p = 1$. For $0 < p < 1$, the MultiLineSearch algorithms of Section 6.2.1 achieve identical results when the positive set is convex, but the non-convexity of these ℓ_p costs precludes the use of the randomized ellipsoid method when the negative class is convex. The ellipsoid method does provide an efficient solution for convex negative sets when $p > 1$ (since these costs are convex). However, for convex positive sets, I show that for $p > 1$ there is no algorithm that can efficiently find an ϵ-$IMAC$ for all $\epsilon > 0$. Moreover, for $p = 2$, I prove that there is no efficient algorithm for finding an ϵ-$IMAC$ for any fixed value of ϵ.

6.4.1 Open Problems in Near-Optimal Evasion

By investigating near-optimal evasion for the convex-inducing classifiers and ℓ_1 costs, I have significantly expanded the extent of the framework established by Lowd and Meek, but there are still a number of interesting unanswered questions about the near-optimal evasion problem. Here I summarize the problems I think are most important and suggest potential directions for pursuing them.

As I shown in this chapter, the current upper bound on the query complexity to achieve near-optimal evasion for the convex positive class is $\mathcal{O}\left(L_\epsilon + \sqrt{L_\epsilon}D\right)$ queries, but the tightest known lower bound is $\mathcal{O}\left(L_\epsilon + D\right)$. Similarly, for the case of convex negative class, the upper bound is given by the randomized ellipsoid approach of Bertsimas and Vempala that finds a near-optimal instance with high probability using $\mathcal{O}^*\left(D^5\right)$ queries (ignoring logarithmic terms). In both cases, there is a gap between the upper and lower bound.

> Question 6.1 Can we find matching upper and lower bounds for evasion algorithms? Is there a deterministic strategy with polynomial query complexity for all convex-inducing classifiers?

The algorithms I present in this chapter built on the machinery of convex optimization over convex sets, which relies on family of classifiers inducing a convex set. However, many interesting classifiers are not convex-inducing classifiers. Currently, the only known result for non-convex-inducing classifiers is due to Lowd and Meek is that linear classifiers on Boolean instance space are 2-$IMAC$ searchable for unweighted ℓ_1 costs. In this case, the classifiers are linear but the integer-valued domains do not have a usual notion of convexity. This raises questions about the extent to which near-optimal evasion is efficient.

> Question 6.2 Are there families larger than the convex-inducing classifiers that are ϵ-$IMAC$ searchable? Are there families outside of the convex-inducing classifiers for which near-optimal evasion is efficient?

A particularly interesting family of classifiers to investigate is the family of support vector machines (SVMs) defined by a particular non-linear kernel. This popular learning

technique can induce non-convex positive and negative sets (depending on its kernel), but it also has a great deal of structure. An SVM classifier can be non-convex in its input space $\mathcal{X}$, but it is always linear in its kernel's Reproducing Kernel Hilbert Space (RKHS). However, optimization within the RKHS is complicated because mapping the cost-balls into the RKHS destroys their structure and querying in the RKHS is non-trivial. However, SVMs also have additional structure that may facilitate near-optimal evasion. For instance, the usual SVM formulation encourages a sparse representation that could be exploited; *i.e.*, in classifiers with few support vectors, the adversary would only need to find these instances to reconstruct the classifier

> QUESTION 6.3 Is some family of SVMs (*e.g.*, with a known kernel) ϵ-*IMAC* searchable for some ϵ? Can an adversary incorporate the structure of a non-convex classifier into the ϵ-*IMAC* search?

In addition to studying particular families of classifiers, it is also of interest to further characterize general properties of a family that lead to efficient search algorithms or preclude their existence. As I showed in this chapter, convexity of the induced sets allows for efficient search for some ℓ_p-costs but not others. Aside from convexity, other properties that describe the shape of the induced sets $\mathcal{X}_f^+$ and $\mathcal{X}_f^-$ could be explored. For instance, one could investigate the family of contiguous-inducing classifiers (*i.e.*, classifiers for which either $\mathcal{X}_f^+$ or $\mathcal{X}_f^-$ is a contiguous, or connected, set). However, it appears that this family is not generally ϵ-*IMAC* searchable since this family includes induced sets with many locally minimal cost regions, which rule out global optimization procedures like the MultiLineSearch or the randomized ellipsoid search. More generally, for families of classifiers that can induce non-contiguous bodies, ϵ-*IMAC* searchability seems impossible to achieve (disconnected components could be arbitrarily close to $\mathbf{x}^A$) unless the classifiers' structure can be exploited. However, even if near-optimal evasion is generally not possible in these cases, perhaps there are subsets of these families that are ϵ-*IMAC* searchable; *e.g.*, as we discuss for SVMs above. Hence, it is important to identify what characteristics make near-optimal evasion inefficient.

> QUESTION 6.4 Are there characteristics of non-convex, contiguous bodies that are indicative of the hardness of the body for near-optimal evasion? Similarly, are there characteristics of non-contiguous bodies that describe their query complexity?

Finally, as discussed in Section 6.1.2, reverse-engineering a classifier (*i.e.*, using membership queries to estimate its decision boundary) is a strictly more difficult problem than the near-optimal evasion problem. Reverse-engineering is sufficient for solving the evasion problem but I show that it is not necessary. Lowd and Meek showed that reverse-engineering linear classifiers is efficient, but here I show that reverse-engineering is strictly more difficult than evasion for convex-inducing classifiers. It is unknown whether there exists a class in between linear and convex-inducing classifiers on which the two tasks are efficient.

> QUESTION 6.5 For what classes of classifiers is reverse-engineering as easy as evasion?

6.4.2 Alternative Evasion Criteria

Here, I suggest variants of near-optimal evasion that generalize or reformulate the problem investigated in this chapter to capture additional aspects of the overall challenge.

6.4.2.1 Incorporating a Covertness Criteria

As mentioned in Section 6.1.2, the near-optimal evasion problem does not require the attacker to be covert in his actions. The primary concern for the adversary is that a defender may detect the probing attack and make it ineffectual. For instance, the MULTILINESEARCH algorithms I present in Section 6.2 are very overt about the attacker's true intention; *i.e.*, because the queries are issued in ℓ_p shells about $\mathbf{x}^A$, it is trivial to infer $\mathbf{x}^A$. The queries issued by the randomized ellipsoid approach in Section 6.2.2 are less overt due to the random walks, but still the queries occur in shrinking cost-balls centered around $\mathbf{x}^A$. The reverse engineering approach of Lowd and Meek [2005b], however, is quite covert. In their approach, all queries are based only on the features of $\mathbf{x}^-$ and a third $\mathbf{x}^+ \in \mathcal{X}_f^+$—$\mathbf{x}^A$ is not used until a ϵ-$IMAC$ is discovered.

> QUESTION 6.6 What covertness criteria are appropriate for a near-optimal evasion problem? Can a defender detect non-discreet probing attacks against a classifier? Can the defender effectively mislead a probing attack by falsely answering suspected queries?

Misleading an adversary is an especially promising direction for future exploration. If probing attacks can be detected, a defender could frustrate the attacker by falsely responding to suspected queries. However, if too many benign points are incorrectly identified as queries, such a defense could degrade the classifier's performance. Thus, strategies to mislead could backfire if an adversary fooled the defender to misclassify legitimate data—yet another security game between the adversary and defender.

6.4.2.2 Additional Information about Training Data Distribution

Consider an adversary that knows the training algorithm and obtains samples drawn from a natural distribution. A few interesting settings include scenarios where the adversary's samples are *i*) a subset of the training data, *ii*) from the same distribution $P_{\mathcal{Z}}$ as the training data, or *iii*) from a perturbation of the training distribution. With these forms of additional information, the adversary could estimate their own classifier $\tilde{f}$ and analyze it offline. Open questions about this variant include:

> QUESTION 6.7 What can be learned from $\tilde{f}$ about f? How can $\tilde{f}$ best be used to guide search? Can the sample data be directly incorporated into ϵ-$IMAC$-search without $\tilde{f}$?

Relationships between between f and $\tilde{f}$ can build on existing results in learning theory. One possibility is to establish bounds on the difference between $MAC(f, A)$ and

$MAC\left(\tilde{f}, A\right)$ in one of the above settings. If, with high probability, the difference is sufficiently small, then a search for an ϵ-$IMAC$ could use $MAC\left(\tilde{f}, A\right)$ to initially lower bound $MAC(f, A)$. This should reduce search complexity since lower bounds on the MAC are typically harder to obtain than upper bounds.

6.4.2.3 Beyond the Membership Oracle

In this scenario, the adversary receives more from the classifier than just a '+'/'−' label. For instance, suppose the classifier is defined as $f(x) = \mathrm{I}[g(x) > 0]$ for some real-valued function g (as is the case for SVMs) and the adversary receives $g(x)$ for every query instead of $f(x)$. If g is linear, the adversary can use $D + 1$ queries and solve a linear regression problem to reverse engineer g. This additional information may also be useful for approximating the support of an SVM.

> QUESTION 6.8 What types of additional feedback may be available to the adversary and how do they impact the query complexity of ϵ-$IMAC$-search?

6.4.2.4 Evading Randomized Classifiers

In this variant of near-optimal evasion, I consider randomized classifiers that generate random responses from a distribution conditioned on the query x. To analyze the query complexity of such a classifier, I first generalize the concept of the MAC to randomized classifiers. I propose the following generalization:

$$RMAC(f, A) = \inf_{x \in \mathcal{X}} \{A(x) + \lambda \mathrm{P}(f(x) = '-')\} \ .$$

Instead of the unknown set $\mathcal{X}_f^-$ in the near-optimal evasion setting, the objective function here contains the term $\mathrm{P}(f(x) = '-')$ that the adversary does not know and must approximate. If f is deterministic , $\mathrm{P}(f(x) = '-') = \mathrm{I}[f(x) = '-']$, this definition is equivalent to Equation (6.2) only if $\lambda \geq MAC(f, A)$ (*e.g.*, $\lambda = A\left(\mathbf{x}^A\right) + 1$ is sufficient); otherwise, a trivial minimizer is $\mathbf{x}^A$. For a randomized classifier, λ balances the cost of an instance with its probability of successful evasion.

> QUESTION 6.9 Given access to the membership oracle only, how difficult is near-optimal evasion of randomized classifiers? Are there families of randomized classifiers that are ϵ-$IMAC$ searchable?

Potential randomized families include classifiers *i*) with fuzzy boundary of width δ around a deterministic boundary, and *ii*) based on the class-conditional densities for a pair of Gaussians, a logistic regression model, or other members of the exponential family. Generally, evasion of randomized classifiers seems to be more difficult than for deterministic classifiers as each query provides limited information about the query probabilities. Based on this argument, Biggio et al. [2010] promote randomized classifiers as a defense against evasion. However, it is not known if randomized classifiers have provable worse query complexities.

6.4.2.5 Evading an Adaptive Classifier

Finally, I consider a classifier that periodically retrains on queries. This variant is a multi-fold game between the attacker and learner, with the adversary now able to issue queries that degrade the learner's performance. Techniques from game-theoretic online learning should be well-suited to this setting [Cesa-Bianchi and Lugosi, 2006].

> Question 6.10 Given a set of adversarial queries (and possibly additional innocuous data) will the learning algorithm converge to the true boundary or can the adversary deceive the learner and evade it simultaneously? If the algorithm does converge, at what rate?

To properly analyze retraining, it is important to have an oracle that labels the points sent by the adversary. If all points sent by the adversary are labeled '+', the classifier may prevent effective evasion, but with a large numbers of false positives due to the adversary queries in $\mathcal{X}_f^-$; this itself constitutes an attack against the learner [Barreno et al., 2010].

6.4.3 Real-World Evasion

While the cost-centric evasion framework presented by Lowd and Meek formalizes the near-optimal evasion problem, it fails to capture some aspects of reality. From the theory of near-optimal evasion, certain classes of learners have been shown to be easy to evade whereas others require a practically infeasible number of queries for evasion to be successful, but real-world adversaries often do not require near-optimal cost evasive instances to be successful; it would suffice if they could find any low-cost instance able to evade the detector. Real-world evasion differs from the near-optimal evasion problem in several ways. Understanding query strategies and the query complexity for a real-world adversary requires overcoming a number of obstacles that were relaxed or ignored in the theoretical version of this problem. Here, I summarize the challenges for real-world evasion.

Real-world near-optimal evasion is harder (*i.e.*, requires more queries) than is suggested by the theory because the theory simplifies the problem faced by the adversary. Even assuming that a real-world adversary can obtain query responses from the classifier, he cannot directly query it in the feature space $\mathcal{X}$. Real-world adversaries must make their queries in the form of real-world objects like email the are subsequently mapped into $\mathcal{X}$ via a feature map. Even if this mapping is known by the adversary, designing an object that maps to a desired query in $\mathcal{X}$ is itself a difficult problem—there may be many objects that map to a single query (*e.g.*, permuting the order of words in a message yields the same unigram representation), and certain portions of $\mathcal{X}$ may not correspond to any real-world object (*e.g.*, for the mapping $x \mapsto \langle x, x^2 \rangle$ no point x can map to $\langle 1, 7 \rangle$).

> Question 6.11 How can the feature mapping be inverted to design real-world instances to map to desired queries? How can query algorithms be adapted for approximate querying?

To adapt to these challenges, I propose an *realistic evasion problem* that weakens several of the assumptions of the theoretical near-optimal evasion problem for studying real-world

evasion techniques. I still assume the adversary does not know f and may not even know the family $\mathcal{F}$; I only assume that the classifier is a deterministic classifier that uniquely maps each instance in $\mathcal{X}$ to $\{'+', negLbl\}$. For a real-world adversary, I require that the adversary send queries that are representable as actual objects in Ω; *e.g.*, emails cannot have 1.7 occurrences of the word "viagra" in a message and IP addresses must have 4 integers between $0--255$. However, I no longer assume that the adversary knows the feature space of the classifier or its feature mapping.

Real-world evasion also differs dramatically from the near-optimal evasion setting in defining an *efficient* classifier. For a real-world adversary, even polynomially-many queries in the dimensionality of the feature space may not reasonable. For instance, if the dimensionality of the feature space is large (*e.g.*, hundreds of thousands of words in unigram models) the adversary may require the number of queries to be sub-linear, $o(D)$, but in the near-optimal evasion problem this is not even possible for linear classifiers. However, real-world adversaries do not need to be provably near-optimal. Near-optimality is a surrogate for adversary's true evasion objective: to use a small number of queries to find a negative instance with acceptably low-cost; *i.e.*, below some maximum cost threshold. This corresponds to an alternative cost function $A'(x) = \max[A(x, \delta)]$ where δ is the maximum allowable cost. Clearly, if a ϵ-$IMAC$ is obtained, either it satisfies this condition or the adversary can cease searching. Thus, ϵ-$IMAC$ searchability is sufficient to achieve the adversary's goal, but the near-optimal evasion problem ignores the maximum cost threshold even though it may allow for the adversary to terminate their search using far fewer queries. To accurately capture real-world evasion with sub-linearly many queries, query algorithms must efficiently use every query to glean essential information about the classifier. Instead of quantifying the query complexity required for a family of classifiers, perhaps it is more important to quantify the query performance of an evasion algorithm for a fixed number of queries based on a target cost.

> QUESTION 6.12 In the real-world evasion setting, what is the worst-case or expected reduction in cost for a query algorithm after making M queries to a classifier $f \in \mathcal{F}$? What is the expected value of each query to the adversary and what is the best query strategy for a fixed number of queries?

The final challenge for real-world evasion is to design algorithms that can thwart attempts to evade the classifier. Promising potential defensive techniques include randomizing the classifier and identifying queries and sending misleading responses to the adversary. I discuss these and other defensive techniques in Chapter 7.1.2.

Chapter 7

Conclusion

Machine learning algorithms are a great potential asset to application developers in enterprise systems, networks, and security domains because these techniques provide the ability to quickly adapt and to find patterns in large diverse data sources. Their potential utility makes analyzing the security implications of these tools a critical task for machine learning researchers and practitioners alike and has spawned a new subfield of research into adversarial learning for security-sensitive domains. The work I have presented in this dissertation significantly advanced the state-of-the-art in this field of study with four primary contributions: a taxonomy for qualifying the security vulnerabilities of a learner, two novel practical attack/defense scenarios for learning in real-world settings, and a generalization of a theoretical paradigm for evading detection of a classifier. However, research in this nascent field has only begun to address obstacles faced in this complex problem domain—many challenges remain. These challenges suggest several new directions for research within both fields of machine learning and computer security. Based on what I have learned in this dissertation, I now provide my outlook on the future of adversarial and secure learning.

Before discussing future directions, I first review the contributions of my dissertation. Above all, I investigated both the practical and theoretical aspects of applying machine learning in security domains. I analyzed the vulnerability of learning systems to adversarial malfeasance; I studied both attacks designed to optimally impact the learning system and attacks constrained by real-world limitations on the adversary's capabilities and information. I further designed defense strategies, which I showed significantly diminish the effect of these attacks. My research focused on learning tasks in virus, spam, and network anomaly detection, but also is broadly applicable across many systems and security domains and has far-reaching implications to any system that incorporates learning. Below, I summarize the contributions of each component of my dissertation followed by a discussion of open problems, lessons learned, and future directions for research.

Framework for Secure Learning The first contribution of my dissertation is a framework for assessing security risks to a learner within a particular context. The basis for this work is a taxonomy of the characteristics of potential attacks, which I jointly developed with my colleagues. From this taxonomy, I developed security games between an attacker and defender which were tailored to the particular type of threat posed by the attacker. The structure of the game was primarily determined by whether or not the attacker could

influence the training data; *i.e.*, either a *Causative* or *Exploratory* attack. The goal of the attacker also contributed to the game by generically specifying the attack function; *i.e.*, whether the attack was an *Integrity* or an *Availability* specified which class of data points are desirable for the adversary and whether the attack is *Targeted* or *Indiscriminate* specifies how broadly the attacker's cost function is concentrated.

Beyond the security games, I also augmented the taxonomy by further exploring the contamination mechanism used by the attacker. I propose a variety of different possible contamination models for an attacker and the role these models play in prior work. Each of these models is appropriate in different scenarios and it is important for an analyst to identify the most appropriate contamination model in their threat assessment. I further demonstrated the use of different contamination models in my subsequent investigation of practical systems.

Causative Attacks against Real-World Learners The second major contribution of my thesis was a practical and theoretical evaluation of two risk minimization procedures in two separate security domains under different contamination models. Within these settings I not only analyzed attacks against real-world systems, I also suggested defense strategies that substantially mitigate the impact of these attacks.

The first system I analyzed in Chapter 4 was the spam filter SpamBayes' learning algorithm. This algorithm is based on a simple probabilistic model for spam and is also used by other spam filtering systems (BogoFilter, Thunderbird's spam filter, and the learning component of SpamAssassin) so the attacks I developed should also be effective against other spam filters but may also be effective against similar learning algorithms used in different domains. Indeed, I demonstrated that the vulnerability of SpamBayes emanates from its modeling assumptions that a message's label depends only on the tokens present in the message and that the tokens are conditionally independent. While these modeling assumptions are not an inherent vulnerability, in this setting conditional independence coupled with the rarity of most tokens and the ability of the adversary to poison large numbers of vulnerable tokens with every attack message makes SpamBayes' learner extremely vulnerable to malicious contamination.

Motivated by the taxonomy of attacks against learners, I designed real-world *Causative* attacks against SpamBayes' learner and demonstrated the effectiveness of these attacks using realistic adversarial control over the training process of SpamBayes. Optimal attacks against SpamBayes caused unreasonably high false positive rates using only a small amount of control of the training process (more than 95% misclassification of ham messages when only 1% of the training data is contaminated). Usenet dictionary attack also effectively use a more realistically limited attack message to cause misclassification of 19% of ham messages with only 1% control over the training messages, rendering SpamBayes unusable in practice. I also show that an informed adversary can successfully target messages. The focused attack changes the classification of the target message virtually 100% of the time with knowledge of only 30% of the target's tokens. Similarly, the pseudospam attack is able to cause nearly 90% of the target spam messages to be labeled as either *unsure* or *ham* with control of less than 10% of the training data.

To combat attacks against SpamBayes, I designed a data sanitization technique called the Reject On Negative Impact (RONI) defense that expunges any message from the training

set if it has an undue negative impact on a calibrated test filter. This technique is a successful defense against dictionary attacks that removed all the variants I tested. However, the RONI defense also has costs: it causes a slight decrease in ham classification, it requires a substantial amount of computation, and it may slow the learning process. Nonetheless, this defense demonstrates that attacks against learners can be detected and prevented.

The second system I presented in Chapter 5 was a PCA-based classifier for detecting anomalous traffic in a backbone network using only volume measurements. This anomaly detection system inherited the vulnerabilities of the underlying PCA algorithm; namely, I demonstrated that PCA's sensitivity to outliers can be exploited by contaminating the training data allowing the adversary to dramatically decrease the detection rate for DoS attacks along a particular target flow.

To counter the PCA-based detector, I studied *Causative Integrity* attacks that poison the training data by adding malicious noise; *i.e.*, spurious traffic sent across the network by compromised nodes that reside within it. This malicious noise is designed to interfere with PCA's subspace estimation procedure. Based on a relaxed objection function, I demonstrated how an adversary can approximate optimal noise using a global view of the traffic patterns in the network. Empirically, I found that by increasing the mean link rate by 10% with *Globally-Informed* chaff traffic, the FNR increased from 3.67% to 38%—a ten-fold increase in misclassification of DoS attacks. Similarly, by only using local link information the attacker is able to mount a more realistic *Add-More-If-Bigger* attack. For this attack, increasing the mean link rate by 10% with *Add-More-If-Bigger* chaff traffic, the FNR increased from 3.67% to 28%—an eight-fold increase in misclassification of DoS attacks. These attacks demonstrate that with sufficient information about network patterns, attacks can mount attacks against the PCA detector that severely compromises its ability to detect future DoS attacks traversing the networking it is monitoring.

I also demonstrated that an alternative robust method for subspace estimation could be used instead to make the resulting DoS detector less susceptible to poisoning attacks. The alternative detector was constructed using a subspace method for robust PCA developed by Croux et al. and a more robust method for estimating the residual cutoff threshold. The resulting ANTIDOTE detector is impacted by poisoning but its performance degrades more gracefully. Under non-poisoned traffic, ANTIDOTE performs nearly as well as PCA, but for all levels of contamination using *Add-More-If-Bigger* chaff traffic, the misclassification rate of ANTIDOTE is approximately half the FNR of the PCA-based solution. Moreover, the average performance of ANTIDOTE is much better than the original detector; it outperforms ordinary PCA for more flows and by a large amount. For multi-week *Boiling Frog* attacks, ANTIDOTE also outperformed PCA and would catch progressively more attack traffic in each subsequent week.

Evasion Attacks The final contribution of my thesis was a generalization of Lowd and Meek's near-optimal evasion framework for quantifying query complexity of classifier evasion to the family of convex-inducing classifiers; *i.e.*, classifiers that partition space into two regions one of which is convex. For the ℓ_p costs, I demonstrated algorithms that can efficiently use polynomially-many queries to find a near-optimal evading instance for any classifier in the convex-inducing classifiers and I showed that for some ℓ_p costs efficient near-optimal evasion cannot be achieved generally for this family of classifiers. Further, the algorithms I present achieve near-optimal evasion without reverse-engineering the classifier

boundary and, in some cases, achieve better asymptotic query complexity than reverse engineering approaches. Further, I show that the near-optimal evasion problem is generally easier than reverse-engineering the classifier's boundary.

My primary contribution from this work is a study of membership query algorithms that efficiently accomplish ϵ-*IMAC* search for convex-inducing classifiers with weighted ℓ_1 costs (*cf.*, Chapter 6.2). When the positive class is convex, I demonstrate efficient techniques that outperform the previous reverse-engineering approaches for linear classifiers. When the negative class is convex, I apply the randomized ellipsoid method introduced by Bertsimas and Vempala to achieve efficient ϵ-*IMAC* search. If the adversary is unaware of which set is convex, he can trivially run both searches to discover an ϵ-*IMAC* with a combined polynomial query complexity; thus, for ℓ_1 costs, the family of convex-inducing classifiers can be efficiently evaded by an adversary; *i.e.*, this family is ϵ-*IMAC* searchable.

Further, I also extended the study of convex-inducing classifiers to general ℓ_p costs (*cf.*, Chapter 6.3). I showed that $\mathcal{F}^{convex}$ is only ϵ-*IMAC* searchable for both positive and negative convexity for any $\epsilon > 0$ if $p = 1$. For $0 < p < 1$, the MultiLineSearch algorithms of Section 6.2.1 achieve identical results when the positive set is convex, but the non-convexity of these ℓ_p costs precludes the use of the randomized ellipsoid method. The ellipsoid method does provide an efficient solution for convex negative sets when $p > 1$ (since these costs are convex). However, for convex positive sets, I show that for $p > 1$ there is no algorithm that can efficiently find an ϵ-*IMAC* for all $\epsilon > 0$. Moreover, for $p = 2$, I prove that there is no efficient algorithm for finding an ϵ-*IMAC* for any fixed value of ϵ.

7.1 Discussion and Open Problems

In the course of my research, I have encountered many challenges and learned important lessons that have given me some insight into the future of the field of adversarial learning in security-sensitive domains. Here I suggest several intriguing research directions for pursuing secure learning. I organize these directions into two topics: *i*) unexplored components of the adversarial game, and *ii*) directions for defensive technologies. Finally, I conclude by enumerating the open problems I suggested throughout this dissertation.

7.1.1 Unexplored Components of the Adversarial Game

As suggested by Chapter 3, adversarial learning and attacks against learning algorithms have recently received a great deal of attention. While many types of attacks have been explored, there are still many elements of this security problem that are relatively unexplored. Here, I summarize the most promising ones for future research.

Research Direction 7.1 (The Role of Measurement and Feature Selection)

As discussed in Chapter 2.2.1, the measurement process and feature selection play an important role in machine learning algorithms that I have ignored in this thesis. However, as suggested in Chapter 3.1, these components of a learning algorithm are also susceptible to attacks. Some prior work has suggested vulnerabilities based on the features used by a learner [*e.g.*, Mahoney and Chan, 2003, Venkataraman et al., 2008, Wagner and Soto, 2002] and others have suggested defenses to particular attacks on the feature set [*e.g.*, Globerson and Roweis, 2006, Sculley et al., 2006], but the role of feature selection remains largely unexplored.

Selecting a set of measurements, is a critical decision in a security-sensitive domain. As has been repeatedly demonstrated [*e.g.*, Wagner and Soto, 2002], irrelevant features can be leveraged by the adversary to cripple the learner's ability to detect malicious instances with little cost to the attacker. For example, in Chapter 4, I showed that tokens unrelated to the spam concept can be used to poison a spam filter. These vulnerabilities require a concerted effort to construct tamper-resistant features, to identify and eliminate features that have been corrupted, and to establish guidelines for practitioners to meet these needs.

Further, feature selection, particularly, may play a pivotal role in the future of secure learning. As discussed in Direction 7.2, these methods can provide secrecy for the learning algorithm and can eliminate irrelevant features. In doing so, feature selection methods may provide a means to gain an advantage against adversaries, but feature selection methods may also be attacked. Exploring these possibilities remains a lucrative research challenge.

Research Direction 7.2 (Secrecy in Learning)

Determining the appropriate degree of secrecy that is feasible for secure machine learning systems is a difficult question but may be critical to providing any strong notion of security. Although, complete transparency seems unreasonable for learning systems if some elements can be effectively kept secret, the source and validity of such secrets remains open to debate:

> QUESTION 7.1 In a learning system, what components of the learning system can be effectively kept secret from an adversary and can keeping these elements secret increase the security of the learner?

Perhaps the most obvious secret is the training data used to create the learned hypothesis. However, in the settings discussed throughout this dissertation, the adversary controls at least some fraction of the input. Further, because learning algorithms generally find patterns in their training data, it is not necessary to exactly reproduce the training data to divulge secrets about the learned hypothesis. In many cases, to approximate the learned hypothesis, the adversary need only have access to a similar dataset.

In Chapter 1.2, I proposed that adversarial learning should adhere to Kerckhoffs' Principle and should assume the adversary is aware of the learning algorithm. This is true, of course, for any open-source or otherwise public software such as the SpamBayes filter I presented in Chapter 4. However, in some settings, keeping the learning algorithm secret may be possible and perhaps prudent. However, even when the algorithm is never revealed, the degree of security provided by algorithmic secrecy is unclear. The adversary may be able to intuit the secret learner; there are a limited number of widespread learning algorithms and only a subset of those are well-suited for a particular task. More importantly, though, the adversary may learn the relevant properties of the learner without knowing the algorithm. As in the near-optimal evasion framework in Chapter 6, the adversary can procure a great deal of information about the learned hypothesis with little information about the training algorithm and hypothesis space.

Instead of keeping the learning algorithm secret, it may be more feasible to hide implementation details such as tuning parameters, kernels (*i.e.*, for SVMs), or the structural model (*e.g.*, for a Bayesian or neural network). Practically speaking, these elements of the algorithm may be easier to hide, but there remains a concern that the adversary does not need to exactly know the algorithm to accomplish his task. These components may themselves be inferred or obviated using techniques such as querying.

Feature selection (as presented in Chapter 2.2.1) could potentially play a role in defending against an adversary by allowing the defender to use dynamic feature selection. In many cases, the goal of the adversary is to construct malicious data instances that are inseparable from innocuous data from the perspective of the learner. However, as the attack occurs, dynamic feature selection could be employed to estimate a new feature mapping $\phi'_{\mathbb{D}}$ that would allow the classifier to continue to separate the classes in spite of the adversary's alterations.

Research Direction 7.3 (Insider Threats in Learning)
I focused exclusively on an external adversary, but an equally potent and often more worrisome threat comes from an internal adversary; *i.e.*, an *insider threat*. Such an adversary can wield far greater powers over the system although they are also often constrained by the greater risk of being exposed by their deeds (and punished for them). Insider threats are a classic concern for traditional computer security, but I am not aware of any research on them in adversarial learning.

> QUESTION 7.2 What threats do a malicious insider pose for a learning system? How do they differ from other threats against the learner and what can the system designer do to prevent them? Can an insider who attacks a learning algorithm be detected and identified?

In the worst-case, an insider adversary with complete control over the learning system can completely destroy the system, but also should be easily identified. More interesting scenarios involve a covert insider. Unlike an external attacker, an insider has access to more privileged information for implementing their attacks. This makes attacks like the focused spam attacks discussed in Chapter 4.3.1.2 more feasible. It also can serve as additional motivation for some of the alternative evasion scenarios discussed in Chapter 6.4.1.

Insiders can also attack by introducing unnatural data directly into the system. For instance, an insider could potentially introduce malicious data points directly into the system rather than having to create real-world data that maps to their desired data points. This capability obviates some of the constraints on real-world attackers discussed in Chapter 6.4.3. An insider could also circumvent other mechanisms meant to protect the learner such as the Reject On Negative Impact (RONI) defense discussed in Chapter 4.4.

Finally, insiders may also possess the capability to introduce data from either the positive or negative class or to change the labels of instances. These capabilities allow an insider to mount attacks that would be otherwise infeasible for an external attacker. For instance, in the spam detection domain, an insider could introduce poison non-spam mail by maliciously mis-labeling training instances. Such an adversary requires a very different threat model than the one used in Chapter 4.2.

7.1.2 Development of Defensive Technologies

The most important challenge remaining for learning in security-sensitive domains is to develop general-purpose secure learning technologies. In Section 3.3.5, I suggested several promising approaches to defend against learning attacks and several secure learners have been proposed [*e.g.*, Dalvi et al., 2004, Globerson and Roweis, 2006, Wang et al., 2006]. However, the development of defenses will inevitably create an arms race, so successful defenses must anticipate potential counterattacks and demonstrate that they are resilient against reasonable threats. With this in mind, the next step is to explore general defenses against larger classes of attack to exemplify trustworthy secure learning.

Research Direction 7.4 (Game-Theoretic Approaches to Secure Learning)

Since suggested by Dalvi et al. [2004], the game-theoretic approach to designing defensive classifiers has rapidly proliferated inspiring several extensions [*e.g.*, Brückner and Scheffer, 2009, Kantarcioglu et al., 2009]. The Dalvi et al. game-theoretic approach is particularly appealing for secure learning because it incorporates the adversary's objective and limitations directly into the classifier's design through an adversarial cost function. However, this cost function is difficult to specify for a real-world adversary and using an inaccurate cost function may again lead to inadvertent blind spots in the classifier. This raises interesting questions:

> Question 7.3 How can a machine learning practitioner design an accurate cost function for a game-theoretic cost-sensitive learning algorithm? How sensitive are these learners to the adversarial cost? Can the cost itself be learned based on observed mistakes?

Game-theoretic learning approaches are especially interesting because they directly incorporate the adversary as part of the learning process. In doing so, they make a number of assumptions about the adversary and his capabilities, but the most dangerous assumption made is that the adversary behaves *rationally* according to their interests. While this assumption seems reasonable, it can cause the learning algorithm to be overly reliant on its model of the adversary. For instance, the original adversary-aware classifier proposed by Dalvi et al. attempts to preemptively detect evasive data but will classify data points as benign if a rational adversary would have altered them; *i.e.*, in this case, the adversary can evade this classifier by simply not changing their behavior. Such strange properties are an undesirable side-effect of the assumption that the adversary is rational, which raises another question:

> Question 7.4 How reliant are adversary-aware classifiers on the assumption that the adversary will behave rationally? Are there game-theoretic approaches that are less dependent on this assumption?

Research Direction 7.5 (Broader Incorporation of Robust Methods)

Currently, choosing a learning method for a particular task is usually based on the structure of application data, the speed of the algorithm in training and prediction, and expected accuracy (often assessed on a static dataset). However, as my research has demonstrated, understanding how an algorithm's performance can change in security-sensitive domains is critical for its success and for widespread adoption in these domains. Designing algorithms to be resilient in these settings is a critical challenge.

Generally, competing against an adversary is a difficult problem and can be computationally intractable. However, the framework of robust statistics as outlined in Chapter 3.5.4.3 addresses the problem of adversarial contamination in training data. This framework provides a number of tools and techniques to construct learners robust against security threats from adversarial contamination. Many classical statistical methods often make strong assumptions that their data is generated by a stationary distribution, but adversaries can defy that assumption. For instance, in Chapter 5, I demonstrated that a robust subspace estimation technique significantly out-performed the original PCA method under adversarial contamination.

Robust statistics augments classical techniques by instead assuming that the data comes from two sources: a known distribution and an unknown adversarial distribution. Under this setting, robust variants exist for parameter estimation, testing, linear models, and other classic statistical techniques. Further, the breakdown point and influence function provide quantitative measurements of robustness, which designers of learning systems can use to evaluate the vulnerability of learners in security-sensitive tasks and select an appropriate algorithm accordingly. However, relatively few learning systems are currently designed explicitly with statistical robustness in mind. I believe, though, that as the field of adversarial learning grows, robustness considerations and techniques will become an increasingly prevalent part of practical learning design. The challenge remains to broadly integrate robust procedures into learning for security-sensitive domains and use them to design learning systems resilient to attacks.

Research Direction 7.6 (Online Learning)

An alternative complementary direction for developing defenses in security-sensitive settings is addressed by the game-theoretic expert aggregation setting described in Chapter 3.6. Recall that in this setting, the learner receives advice from a set of experts and makes a prediction by weighing the experts' advice based on their past performance. Techniques for learning within this framework have been developed to perform well with respect to the best expert in hindsight. A challenge that remains is designing sets of experts that together can better meet a security objective. Namely,

> QUESTION 7.5 How can one design a set of experts (learners) so that their aggregate is resilient to attacks in the online learning framework?

Ideally, even if the experts may be individually vulnerable, they are difficult to attack as a group. I informally refer to such a set of experts as being *orthogonal*. Orthogonal learners have several advantages in a security sensitive environment. They allow us to combine learners designed to capture different aspects of the task. These learners may use different feature sets and different learning algorithms to reduce common vulnerabilities; *e.g.*, making them more difficult to reverse engineer. Finally, online expert aggregation techniques are flexible: existing experts can be altered or new ones can be added to the system whenever new vulnerabilities in the system are identified.

To properly design a system of orthogonal experts for secure learning, the designer must first assess the vulnerability of several candidate learners. With that analysis, he should then choose a base set of learners and sets of features for them to learn on. Finally, as the aggregate predictor matures, the security analyst should identify new security threats and patch the learners appropriately. This patching could be done by adjusting the algorithms, changing their feature sets, or even adding new learners to the aggregate. Perhaps this process could itself be automated or learned.

7.2 Review of Open Problems

Many exciting challenges remain in the field of adversarial learning in security-sensitive domains. Here I recount the open questions I suggested throughout this dissertation.

Problems from Chapter 5

Problems from Chapter 6

7.3 Concluding Remarks

The field of adversarial learning in security-sensitive domains is a new and rapidly expanding sub-discipline that holds a number of interesting research topics for researchers in both machine learning and computer security. My dissertation research has both significantly impacted this community and highlighted several important lessons. First, to design effective learning systems, practitioners must follow the principle of proactive design as discussed in Chapter 1.2. To avoid security pitfalls, designers must develop reasonable threat models for potential adversaries and develop learning systems to meet their desired security requirements. At the same time, machine learning designers should promote the security properties of their algorithms in addition to other traditional metrics of performance.

A second lesson that has re-emerged throughout this dissertation is that there are inherent trade-offs between a learner's performance on regular data and its resilience to attacks.

Understanding these trade-offs is important not only for security applications but also for understanding how learners behave in any non-ideal setting.

Finally, throughout this dissertation, I suggested a number of promising approaches toward secure learning, but a clear picture of what is required for secure learning has yet to emerge. Each of the approaches I discussed are founded in game theory but have different benefits: the adversary-aware classifiers directly incorporate the threat model into their learning procedure, the robust statistics framework provides procedures that are generally resilient against any form of contamination, and the expert aggregation setting constructs classifiers that can do nearly as well as the best expert in hindsight. However, by themselves, none of these form a complete solution for secure learning. Integrating these different approaches or developing a new approach remains as the most important challenge for this field.

List of Symbols

$A(\cdot)$ — The adversary's cost function on $\mathcal{X}$ (see Chapter 6.1.1). 130–137, 139, 141, 143, 145, 152, 154, 157, 158, 220, 225

$\mathbb{D}$ — A set of data points (see also: dataset). 27–30

— N — The number of data points in the training dataset use by a learning algorithm; *i.e.*, $N \triangleq \left|\mathbb{D}^{(\text{train})}\right|$. 21, 22, 25, 27–31, 39, 41, 42, 46, 48, 50, 52, 54, 185

— $\mathbb{D}^{(\text{train})}$ — A dataset used by a training algorithm to construct or select a classifier (see also: dataset). 25, 29, 30, 39, 41–43, 48, 50, 63, 76, 87

— $\mathbb{D}^{(\text{eval})}$ — A dataset used to evaluate a classifier (see also: dataset). 25, 29, 31, 39, 41–43, 47, 48, 50, 87

$\triangleq$ — Symbol used to provide a definition; *e.g.*, $\pi \triangleq 3.14159\ldots$. 21–24, 27, 28, 30, 55, 57, 63, 64, 96, 98, 99, 101, 107, 112, 130, 132, 204, 207, 209, 210, 212, 213

ϵ-$IMAC$ — The set of objects in $\mathcal{X}_f^-$ within a cost of $1+\epsilon$ of the MAC, or any of the members of this set (see also: $MAC(f, A)$). 130–132, 134–139, 141, 144, 145, 148, 153–158, 160, 171, 217, 220–223, 225

$f(\cdot)$ — The classifier function or hypothesis learned by a training procedure $H^{(N)}$ from the dataset $\mathbb{D}^{(\text{train})}$ (see also: classifier). 25, 28–31, 41, 42, 48, 50, 52, 76, 98, 129–135, 137, 140, 142, 144, 145, 157, 158, 160, 171, 172, 175, 220, 221, 225

L_ϵ — The number of steps required by a binary search to achieve ϵ-optimality (see Chapter 6.1.3). 131, 135, 137, 139, 141, 143–145, 148, 151, 155, 217–219

$MAC(f, A)$ — The largest lower bound on the adversary's cost A over $\mathcal{X}_f^-$ (see also: Equation 6.2). 130–133, 135, 137–139, 141, 144, 145, 152, 153, 158, 175, 220–223, 225

$\aleph$ — The set of natural numbers, $\{1, 2, 3, \ldots\}$. 22, 96, 207

$\|\cdot\|$ — A non-negative function defined on a vector space that is positive homogeneous and obeys the triangle inequality (see also: norm). 22, 23, 104, 105, 107, 110, 112, 130, 149, 150, 223

— ℓ_p $(p > 0)$ — A norm on a multi-dimensional real-value space defined in Chapter 2.1 by Equation (2.1) and denoted by $\|\cdot\|_p$. 7, 18–20, 128, 130, 134–137, 139, 141, 143–146, 148, 149, 151–157, 163, 164, 199, 203–205, 222, 223, 225

— $m_{\mathbb{C}}(\cdot)$ — A function that defines a distance metric for a convex set $\mathbb{C}$ relative to some central element $\mathbf{x}^{(c)}$ in the interior of $\mathbb{C}$ (see also: Minkowski metric). 135, 136

$N^{(h)}$ — The total number of ham messages in the training dataset. 63, 64, 67, 208–211

$n_j^{(h)}$ — The number of occurences of the j^{th} token in training ham messages. 63, 64, 67, 208–211

$N^{(s)}$ — The total number of spam messages in the training dataset. 63, 64, 67, 75, 208–211

$n_j^{(s)}$ — The number of occurrences of the j^{th} token in training spam messages. 63, 64, 67, 75, 208–211

$\mathbf{Q}$ — The matrix of network flow data. 96, 110

$\mathbf{R}$ — The routing matrix that describes the links used to route each OD flow. 96, 100, 110

$\Re$ — The set of all real numbers. 22–24, 27–29, 31, 101

— $\Re^{0+}$ — The set of all real number greater than or equal to zero. 22, 30, 130, 136, 207

— $\Re^{+}$ — The set of all real numbers greater than zero. 22, 23, 31, 141, 220

— $\Re^{D}$ — The D-dimensional real-valued space. 22, 24, 28, 97, 98, 101, 105, 129, 141, 149, 220, 221, 224

$\mathbf{x}$ — A data point from the input space $\mathcal{X}$ (see also data point). 22, 27, 28, 96–98, 100, 104–106, 128, 130–132, 134–136, 138, 147, 149

Glossary

ACRE-learnable The original framework proposed by Lowd and Meek [2005b] for quantifying the query complexity of a family of classifiers; see also, near-optimal evasion problem. 48

action In the context of a learning algorithm, a response or decision made by the learner based on its predicted state of the system. 25

additive gap ($G^{(+)}$) The additive difference between the estimated optimum $\hat{C}$ and the global optimum C^* as measured by the difference between these two quantities: $\hat{C} - C^*$. When the global optimum is not known, this gap refers to the difference between the estimated optimum and a *lower bound* on the global optimum. 133

additive optimality A form of approximate optimality where the estimated optimum $\hat{C}$ is compared to global optimum C^* using the difference $\hat{C} - C^*$; η-additive optimality is achieved when this difference is less than or equal to η. 132

adversarial learning Any learning problem where the learning agent faces an adversarial opponent who wants the learner to fail in some way. Specifically, in this dissertation, I consider adversarial learning in security-sensitive domains. 16, 18, 19, 54

anomaly detection The task of identifying anomalies within a set of data. 30, 32, 93

attacker In the learning games introduced in Chapter 3, the attacker is the malicious player who is trying to defeat the learner. 30, 38

batch training Training in which all training data is examined in batch by the learning algorithm to select its hypothesis, f. 29, 49

beta distribution A continuous probability distribution with support on $(0, 1)$ parameterized by $\alpha \in \Re^+$ and $\beta \in \Re^+$ that has a probability density function given by $\frac{x^{\alpha-1}(1-x)^{\beta-1}}{\mathrm{B}(\alpha,\beta)}$. 65

beta function ($\mathrm{B}(\alpha, \beta)$) A two-parameter function defined by the definite integral $\mathrm{B}(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}\,\mathrm{d}t$ for parameters $\alpha > 0$ and $\beta > 0$. 67

blind spot a class of miscreant activity that fails to be correctly detected by a detector; *i.e.*, false positives. 16, 20, 58, 127, 183

breakdown point (ϵ^*) Non-formally, it is the largest fraction of malicious data that an estimator can tolerate before the adversary can use the malicious data to arbitrarily change the estimator. The breakdown point of a procedure is one measure of its robustness. 54, 55, 104, 169

classification A learning problem in which the learner is tasked with predicting a response in its response space $\mathcal{Y}$ given an input x from its input space $\mathcal{X}$. In a classification problem, the learned hypothesis is referred to as a classifier. The common case when the response case is boolean or $\{0, 1\}$ is referred to as binary classification. 27, 30, 180, 185

binary classification A classification learning problem where the response space $\mathcal{Y}$ is a set of only two elements; *e.g.*, $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{'+', '-'\}$. 30, 31, 179, 185

classifier (f) A function $f : \mathcal{X} \to \mathcal{Y}$ that predicts a response variable based on a data point $\mathbf{x} \in \mathcal{X}$. In classification, the classifier is selected from the space $\mathcal{F}$ based on a labeled dataset $\mathbb{D}^{(\text{train})}$; *e.g.*, in the empirical risk minimization framework. 30, 175, 179, 180

convex combination A linear combination $\sum_i \alpha_i \cdot \mathbf{x}^{(i)}$ of the vectors $\left\{\mathbf{x}^{(i)}\right\}$ where the coefficients satisfy $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$. 22

convex optimization program A mathematical optimization problem in which a convex function is minimized over a convex set. 136

convex set A set $\mathbb{A}$ is convex if for any pair of objects $a, b \in \mathbb{A}$, all convex combinations of a and b are also in $\mathbb{A}$; *i.e.*, $\alpha a + (1 - \alpha) b \in \mathbb{A}$ for all $\alpha \in [0, 1]$. 22

convex-inducing classifier A binary classifier f for which either $\mathcal{X}_f^+$ or $\mathcal{X}_f^-$ is a convex set. 7, 19, 20, 58, 128, 129, 132, 135–137, 143, 148, 154, 155, 163, 164, 220, 221

cost function A function that describes the cost incurred in a game by a player (the adversary or learner) for their actions. In this dissertation, the cost for the learner is a loss function based solely on the learners predictions whereas the cost for the adversary may also be data dependent. 36

covering number The minimum number of balls need to cover an object and hence, a measure of the objects complexity. 131

data A set of observations about the state of a system. 25

data collection The process of collecting a set of observations about the system that comprise a dataset. 27, 46, 181

data point ($\mathbf{x}$) An element of a dataset that is a member of $\mathcal{X}$. 16, 27, 28, 30, 31, 176, 180, 181, 183

data sanitization The process of removing anomalous data from a dataset prior to training on it. 19, 58

dataset ($\mathbb{D}$) An indexed set of data points denoted by $\mathbb{D}$. 27, 29, 30, 175, 180, 182

defender In the learning games introduced in Chapter 3, the defender is learning agent who plays against an attacker. If the learning agent is able to achieve its security goals in the game, it has achieved secure learning. 30, 38

degree of security the level of security expected against an adversary with a certain set of objectives, capabilities, and incentives based on a threat model. 9

denial-of-service attack (DoS) An attack that disrupts normal activity within a system. 6, 19, 45, 52, 93, 96–102, 105, 109, 110, 112, 122, 163

dictionary attack A *Causative Availability* attack against SpamBayes, in which attack messages contain an entire dictionary of tokens to be corrupted. 68, 69

dispersion The notion of the spread or variance of a random variable (also known as the scale or deviation). Common estimators of dispersion include the standard deviation and the median absolute deviation. 94, 95, 104–106

distributional robustness A notion of robustness against deviations from the distribution assumed by a statistical model; *e.g.*, outliers. 95

empirical risk minimization The learning principle of selecting a hypothesis that minimizes the empirical risk over the training data. 30, 180

empty set The set containing no objects. 21

expert An agent that can make predictions or give advice that is used to create a composite predictor based on the advice received from a set of experts. 56

explanatory variable An observed quantity that is used to predict an unobservable response variable. 27

false negative An erroneous prediction that a positive instance is negative. 31, 35, 93

false negative rate The frequency at which a predictor makes false negatives. In machine learning and statistics, this is a common performance measure for assessing a predictor along with the false positive rate. 31, 61, 93, 109, 181, 186

false positive An erroneous prediction that a negative instance is positive. 31, 179

false positive rate The frequency at which a predictor makes false positives. In machine learning and statistics, this is a common performance measure for assessing a predictor along with the false negative rate. 31, 61, 93, 109, 181

feature An element of a data point; typically a particular measurement of the overall object that the data point represents. 27

feature deletion attack An attack proposed by Globerson and Roweis [2006] in which the adversary first causes a learning agent to associate intrusion instances with irrelevant features and subsequently removes these spurious features from his intrusion instances to evade detection. 47

feature selection The second phase of data collection in which the data are mapped to an alternative space $\hat{\mathcal{X}}$ to select the most relevant representation of the data for the learning task. This dissertation does not distinguish between the feature selection and measurement phases; instead they are considered to be a single step and $\mathcal{X}$ is used in place of $\hat{\mathcal{X}}$. 28, 181

feature selection map (ϕ) The (data-dependent) function used by feature selection to map from the original input space $\mathcal{X}$ to a second feature space $\hat{\mathcal{X}}$ of the features most relevant for the subsequent learning task. 28, 46, 159

Gaussian distribution ($\mathrm{N}(\mu, \sigma)$) A continuous probability distribution with support on $\Re$ parameterized by a center $\mu \in \Re$ and a scale $\sigma \in \Re^+$ that has a probability density function given by $\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$. 29

good word attack A spam attack studied by Wittel and Wu [2004] and Lowd and Meek [2005a], in which the spammer adds words associated with non-spam messages to their spam in order to evade a spam filter. More generally, any attack where an adversary adds features to make intrusion instances appear to be normal instances. 44

gross-error model ($\mathcal{P}_\epsilon(F_\mathcal{Z})$) A family of distributions about the known distribution $F_\mathcal{Z}$ parameterized by the fraction of contamination ϵ that combine $F_\mathcal{Z}$ with a fraction ϵ of contamination from distributions $H_\mathcal{Z} \in \mathcal{P}_\mathcal{Z}$. 55

gross-error sensitivity The supremum, or smallest upper bound, on the magnitude of the influence function for an estimator; this serves as a quantitative measure of a procedure or estimator. 55

hypothesis (f) A function f mapping from the data space $\mathcal{X}$ to the response space $\mathcal{Y}$. The task for a learner is to select a hypothesis from its hypothesis space to best predict the response variables based on the input variables. 25, 28–31, 179, 181, 184, 185

hypothesis space ($\mathcal{F}$) The set of all possible hypotheses, f, that are supported by the learning model. While this space is often infinite, it is indexed by a parameter $\boldsymbol{\theta}$ that maps to each hypothesis in the space. 28–30, 181, 185

index set A set $\mathbb{I}$ that is used as an index to the members of another set $\mathbb{X}$ such that there is a mapping from each element of $\mathbb{I}$ to a unique element of $\mathbb{X}$. 22

indicator function The function $\mathrm{I}[\cdot]$ that is 1 when its argument is true and is 0 otherwise. 21

inductive bias A set of (implicit) assumptions used in inductive learning to bias generalizations from a set of observations. 25

inductive learning A task where the learner generalizes a pattern from training examples; *e.g.*, finding a linear combination of features that empirically discriminates between positive and negative data points. 25

influence function ($\mathrm{IF}(z; H, F_{\mathcal{Z}})$) A functional used extensively in robust statistics that quantifies the impact of an infinitesimal point contamination at z on an asymptotic estimator H on distribution $F_{\mathcal{Z}}$; see Chapter 3.5.4.3. 54, 55, 169, 181

input space ($\mathcal{X}$) The space of all data points. 27, 176, 177, 179

intrusion detection system A detector that is designed to identify suspicious activity that is indicative of illegitimate intrusions. Typically these systems are either host-based or network-based detectors. 35

intrusion instance A data point that corresponds to an illegitimate activity. The goal of malfeasance detection is to properly identify normal and intrusion instances and prevent the intrusion instances from achieving their intended objective. 30, 181, 184

intrusion prevention system A system tasked with detecting intrusions and taking automatic actions to prevent detected intrusions from succeeding. 35

iterated game In game theory, a game in which players choose moves in a series of repetitions of the game. 56, 184

label A special aspect of the world that is to be predicted in a classification problem or past examples of this quantity associated with a set of data points that are jointly used to train the predictor. 27, 30, 177

labeled dataset A dataset in which each data point has an associated label. 27, 180

learner An agent or algorithm that performs actions or makes predictions based on past experiences or examples of how to properly perform its task. When presented with new examples, the learner should adapt according to a measure of its performance. 28, 181

learning algorithm Any algorithm that adapts to a task based on past experiences of the task and a performance measure to assess its mistakes. 29

loss function A function, commonly used in statistical learning, that assesses the penalty incurred by a learner for making a particular prediction/action compared to the best or correct one according to the *true* state of the world; *e.g.*, the squared loss for real-valued prediction is given by $L(y, \hat{y}) \triangleq (\hat{y} - y)^2$. 25, 30, 180

machine learning A scientific discipline that investigates algorithms that adapt their behavior based on past experiences and observations. As stated by Mitchell [1997], "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". 25

malfeasance detection The task of detecting some particular form of illegitimate activity; *e.g.*, virus, spam, intrusion, or fraud detection. 2, 182

measurement An object mapped from the space of real-world object to the data representation used by a learning algorithm. 27

measurement map A description of the process that creates a measurement based on the observations and properties of a real world object. 27

median absolute deviation A robust estimator for dispersion defined by Equation (5.5), which attains the highest possible breakdown point of 50% and is the most robust M-estimator for dispersion. 95

membership query A query sent to an oracle to determine set membership for some set defined by the oracle's responses. 130, 154

mimicry attack An attack where the attacker tries to disguise malicious activity to appear to be normal. 47

minimal adversarial cost (MAC) The smallest adversarial cost A that can be obtained for instances in the negative class $\mathcal{X}_f^-$ of a deterministic classifier f. 130

Minkowski metric A distance metric for the convex set $\mathbb{C}$ which is defined relative to a point $\mathbf{x}^{(c)}$ in the interior of the set. 135, 176

multiplicative gap ($G^{(*)}$) The multiplicative difference between the estimated optimum $\hat{C}$ and the global optimum C^* as measured by the ratio between these two quantities: $\frac{\hat{C}}{C^*}$. When the global optimum is not known, this gap refers to the ratio between the estimated optimum and a *lower bound* on the global optimum. 133

multiplicative optimality A form of approximate optimality where the estimated optimum $\hat{C}$ is compared to global optimum C^* using the ratio $\frac{\hat{C}}{C^*}$; ϵ-multiplicative optimality is achieved when this ratio is less than or equal to $1 + \epsilon$. 132

near-isotropic A set or body that is nearly round as defined by Equation 6.11. 147, 148

near-optimal evasion problem A framework for measuring the difficulty for an adversary to find blind spots in a classifier using a probing attack with few queries. A family of classifiers is considered difficult to evade if there is no efficient query-based algorithm for finding near-optimal instances; see Chapter 6. 7, 48, 127, 131, 179, 184

negative class The set of data points that are classified as negative by the classifier f (denoted by $\mathcal{X}_f^-$). 30, 135–137, 148, 154, 155, 177, 183

norm ($\|\cdot\|$) A non-negative function on a vector space $\mathcal{X}$ that is zero only for the zero vector $\mathbf{0} \in \mathcal{X}$, is positive homogeneous, and obeys the triangle inequality. 22, 176

normal instance A data point that represents normal (allowable) activity such as a regular email message. 30, 181, 184

obfuscation Any method used by adversaries (particularly spammers) to conceal their malfeasance. 5, 7, 11, 43, 44, 47, 68, 74

Ockham's Razor An assumption that the simplest hypothesis is probably the correct one. 25

OD flow volume anomaly An unusual traffic pattern in an OD flow between two points-of-presence (PoPs) in a communication network; *e.g.*, a DoS attack. 96

one-class support vector machine A formulation of the support vector machine used for anomaly detection. 14

one-shot game In game theory, any game in which players each make only a single move. 55

online training Training in which data points from the training dataset arrive sequentially.

Often, online training consists of sequential prediction followed by re-training as described in Chapter 3.6. 29

overfitting A phenomenon in which a learned hypothesis fails to generalize to test data; *i.e.*, it poorly predicts new data items drawn from the same distribution. Typically this occurs because the model has too much complexity for its training data and captures random fluctuations in it rather than the underlying relationships. Note, this phenomenon is distinct from non-stationarity; *e.g.*, distribution shift. 31, 184

PCA Evasion Problem A problem discussed in Chapter 5 in which the attacker attempts to send DoS attacks that evade detection by a PCA subspace-based detector as proposed by Lakhina et al. [2004b]. 100

performance measure A function used to assess the predictions made by or actions taken by a learning agent. 30, 181, 182, 185

polymorphic blending attack Attacks proposed by Fogla and Lee [2006] that use encryption techniques to make intrusion instances indistinguishable from normal instances. 43

positive class ($\mathcal{X}_f^+$) The set of data points that are classified as positive by the classifier f (denoted by $\mathcal{X}_f^+$). 30, 130, 135–137, 148, 154, 177, 220–222, 225

positive homogeneous function Any function p on a vector space $\mathcal{X}$ that satisfies $p(a\mathbf{x}) = |a|\, p(\mathbf{x})$ for all $a \in \Re$ and $\mathbf{x} \in \mathcal{X}$. 139, 176, 183

prediction The task of predicting an unobserved quantity about the state of a system based on observable information about the system's state and past experience. 29

prior distribution A distribution on the parameters of a model that reflects information or assumptions about the model formed before obtaining empirical data about it. 63, 210

probably approximately correct A learning framework introduced by Valiant [1984] in which the goal of the learner is to select a hypothesis that achieve a low training error with high probability. 53

probing attack An attack which uses queries to discern hidden information about a system that could expose its weaknesses; see near-optimal evasion problem. 42, 183

query A questionb posed to an oracle; in an adversarial learning setting, queries can be used to infer hidden information about a learning agent. 42, 45, 48, 128, 129, 131–134, 136, 137, 139, 141, 143, 144, 146, 148, 152, 153, 155, 158–160

regret The difference in loss incurred by a composite predictor and the loss of an expert used by the composite in forming its predictions. 56, 57

- **cumulative regret** ($R^{(m)}$) The total regret received for the m^{th} expert over the course of K rounds of an iterated game. 57, 184
- **instantaneous regret** ($r^{(k,m)}$) The difference in loss between the composite predictor and the m^{th} expert in the k^{th} round of the game. 56
- **worst-case regret** (R^*) The maximum cumulative regret for a set of M experts. 57, 184

regret minimization procedure A learning paradigm in which the learner dynamically re-weighs advice from a set of experts based on their past performance so that the resulting combined predictor has a small worst-case regret; *i.e.*, it predicts almost as well as the best expert in hindsight. 57

regularization The process of providing additional information or constraints in a learning problem to solve an ill-posed problem or to prevent overfitting, typically by penal-

izing hypothesis complexity or introducing a prior distribution. Regularization techniques include smoothness constraints, bounds on the norm of the hypothesis, $\|f\|$, and prior distributions on parameters. 31

residual rate A statistic which measures the change in the size of the residual caused by adding a single unit of traffic volume into the network along a particular OD flow. Alternatively, it can be thought of as a measure of how closely a subspace aligns with the flow's vector. 110

response space ($\mathcal{Y}$) The space of values for the response variables; in classification this is a finite set of categories and in binary classification it is $\{'+', '-'\}$. 27, 30, 177, 179, 180

response variable An unobserved quantity that is to be predicted based on observable explanatory variables. 27, 180, 185

risk ($R(P_{\mathcal{Z}}, f)$) The expected loss of a decision procedure f with respect to data drawn from the distribution $P_{\mathcal{Z}}$. 30

robust statistics The study and design of statistical procedures that are resilient to small deviations from the assumed underlying statistical model; *e.g.*, outliers. 53

scale invariant A property that does not change when the space is scaled by a constant factor. 134

secure learning The ability of a learning agent to achieve its security goals in spite of the presence of an adversary who tries to prevent it from doing so. 3, 180

security goal Any objective that a system needs to achieve to ensure the security of the system and/or its users. 35

security-sensitive domain A task or problem domain in which malicious entities have a motivation and a means to disrupt the normal operation of system. In the context of glsadversarial learning, these are problems where and adversary wants to mislead or evade a learning algorithm. 1–3, 33, 35

set A group of objects. 21

set indicator function The function $\mathrm{I}_{\mathbb{X}}[\cdot]$ associated with the set $\mathbb{X}$ that is 1 for any $x \in \mathbb{X}$ and is 0 otherwise. 21

shift invariant A property that does not change when the space is shifted by a constant amount. 134

stationarity A stochastic process in which a sequence of observations are all drawn from the *same* distribution. Also, in machine learning, it is often assumed that the training and evaluation data are both drawn from the same distribution—I refer to this as an assumption of stationarity. 25, 35

support vector machine A family of (non-linear) learning algorithms that find a maximally separating hyperplane in a high-dimensional space known as its Reproducing Kernel Hilbert Space (RKHS). The kernel function allows the SVM to compute inner products in that space without explicitly mapping the data into the RKHS.. 42

threat model A description of an adversary's incentives, capabilities and limitations. 9, 35, 180

training The process of using a training dataset $\mathbb{D}^{(\text{train})}$ to choose a hypothesis f from among a hypothesis space, $\mathcal{F}$. 29, 179, 183

training algorithm ($H^{(N)}$) An algorithm that selects a classifier to optimize a performance

measure for a training dataset; also known as an estimating procedure or learning algorithm. 29

true positive rate The frequency for which a predictor correctly classifies positive instances. This is a common measure of a predictor's performance and is one minus the false negative rate. 109

unfavorable evaluation distribution A distribution introduced by the adversary during the evaluation phase to defeat the learner's ability to make correct predictions; this is also referred to as *distributional drift*. 46

VC-dimension The VC or Vapnik-Chervonenkis dimension is a measure of the complexity of a family of classifiers, which is defined as the cardinality of the largest set of data points that can be shattered by the classifiers. 131

vector An element in a vector space for which vector addition and scalar multiplication are defined. 22, 186

vector space A set of objects (vectors) that can be added or multiplied by a scalar; *i.e.*, the space is closed under vector addition and scalar multiplication operations that obey associativity, commutativity, and distributivity and has an additive and multiplicative identity as well as additive inverses. 22, 176, 183, 184, 186

virus detection system A detector tasked with identifying potential computer viruses. 35

Bibliography

Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A. Maltz, and Ming Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, pages 13–24, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-713-1.

Marco Barreno. *Evaluating the Security of Machine Learning Algorithms.* PhD thesis, University of California at Berkeley, May 2008.

Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 16–25, New York, NY, USA, 2006. ACM. ISBN 1-59593-272-0.

Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, November 2010.

Dimitris Bertsimas and Santosh Vempala. Solving convex programs by random walks. *Journal of the ACM*, 51(4):540–556, 2004.

Battista Biggio, Giorgio Fumera, and Fabio Roli. Multiple classifier systems under attack. In Neamat El Gayar; Josef Kittler; Fabio Roli, editor, *Proceedings of the 9th International Workshop on Multiple Classifier Systems (MCS)*, volume 5997, pages 74–83, Cairo, Egypt, July 2010. Springer. ISBN 978-3-642-12126-5.

Patrick Billingsley. *Probability and Measure.* Wiley, New York, NY, USA, 3rd edition, 1995. ISBN 978-0471007104.

Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006. ISBN 0-387-31073-8.

Peter Bodík, Rean Griffith, Charles Sutton, Armando Fox, Michael I. Jordan, and David A. Patterson. Statistical machine learning makes automatic control practical for internet datacenters. In *Proceedings of the Workshop on Hot topics in cloud computing (HotCloud)*, pages 12–17, Berkeley, CA, USA, 2009. USENIX Association.

Peter Bodík, Armando Fox, Michael J. Franklin, Michael I. Jordan, and David A. Patterson. Characterizing, modeling, and generating workload spikes for stateful services. In *Proceedings of the 1st ACM symposium on Cloud computing (SoCC)*, pages 241–252, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0036-0.

Richard J. Bolton and David J. Hand. Statistical fraud detection: A review. *Journal of Statistical Science*, 17(3):235–255, 2002.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. ISBN 978-0-521-83378-3.

Daniela Brauckhoff, Kavé Salamatian, and Martin May. Applying PCA for traffic anomaly detection: Problems and solutions. In *Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM)*, pages 2866–2870. IEEE, April 2009.

Michael Brückner and Tobias Scheffer. Nash equilibria of static prediction games. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 22, pages 171–179. MIT Press, 2009.

Nicolò Cesa-Bianchi and Gábor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, New York, NY, USA, 2006. ISBN 0-521-84108-9.

Yu-Chung Cheng, Mikhail Afanasyev, Patrick Verkaik, Péter Benkö, Jennifer Chiang, Alex C. Snoeren, Stefan Savage, and Geoffrey M. Voelker. Automating cross-layer diagnosis of enterprise wireless networks. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 25–36, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-713-1.

Andreas Christmann and Ingo Steinwart. On robustness properties of convex risk minimization methods for pattern recognition. *Journal of Machine Learning Research (JMLR)*, 5: 1007–1034, 2004. ISSN 1533-7928.

Simon P. Chung and Aloysius K. Mok. Allergy attack against automatic signature generation. In Diego Zamboni and Christopher Krügel, editors, *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 4219 of *Lecture Notes in Computer Science*, pages 61–80. Springer, September 2006. ISBN 3-540-39723-X.

Simon P. Chung and Aloysius K. Mok. Advanced allergy attacks: Does a corpus really help? In Christopher Krügel, Richard Lippmann, and Andrew Clark, editors, *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 4637 of *Lecture Notes in Computer Science*, pages 236–255. Springer, September 2007. ISBN 978-3-540-74319-4.

Gordon Cormack and Thomas Lynam. Spam corpus creation for TREC. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, July 2005.

Christophe Croux and Anne Ruiz-Gazen. High breakdown estimators for principal components: the projection-pursuit approach revisited. *Journal of Multivariate Analysis*, 95 (1):206–226, July 2005. ISSN 0047-259X.

Christophe Croux, Peter Filzmoser, and M. Rosario Oliveira. Algorithms for projection-pursuit robust principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 87(2):218–225, 2007.

Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 99–108, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-888-1.

Susan J. Devlin, Ramanathan Gnanadesikan, and Jon R. Kettenring. Robust estimation of dispersion matrices and principal components. *Journal of the American Statistical Association*, 76:354–362, 1981.

Mark Dredze, Reuven Gevaryahu, and Ari Elias-Bachrach. Learning fast classifiers for image spam. In *Proceedings of the 4th Conference on Email and Anti-Spam (CEAS)*, August 2007.

Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Salvatore J. Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In *Data Mining for Security Applications*. Kluwer, 2002.

Ronald A. Fisher. Question 14: Combining independent tests of significance. *American Statistician*, 2(5):30–31, 1948.

Prahlad Fogla and Wenke Lee. Evading network anomaly detection systems: Formal reasoning and practical techniques. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)*, pages 59–68, New York, NY, USA, 2006. ACM. ISBN 1-59593-518-5.

Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for unix processes. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 120–128, Los Alamitos, CA, USA, May 1996. IEEE Computer Society. ISBN 0-8186-7417-2.

Amir Globerson and Sam Roweis. Nightmare at test time: Robust learning by feature deletion. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 353–360, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2.

Paul Graham. A plan for spam. `http://www.paulgraham.com/spam.html`, August 2002.

Frank R. Hampel, Elvezio M. Ronchetti, Peter J. Rousseeuw, and Werner A. Stahel. *Robust Statistics: The Approach Based on Influence Functions*. Probability and Mathematical Statistics. John Wiley and Sons, New York, NY, USA, 1986. ISBN 0-471-73577-9.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2003. ISBN 978-0387-95284-0.

Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998. ISSN 0926-227X.

Ola Hössjer and Christophe Croux. Generalizing univariate signed rank statistics for testing and estimating a multivariate location parameter. *Journal of Nonparametric Statistics*, 4(3):293–308, 1995.

Ling Huang, XuanLong Nguyen, Minos Garofalakis, Michael I. Jordan, Anthony Joseph, and Nina Taft. In-network PCA and anomaly detection. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19 (NIPS)*, pages 617–624, Cambridge, MA, USA, 2007. MIT Press.

Peter J. Huber. *Robust Statistics*. Probability and Mathematical Statistics. John Wiley and Sons, New York, NY, USA, 1981. ISBN 0-471-41805-6.

J. Edward Jackson and Govind S. Mudholkar. Control procedures for residuals associated with principal component analysis. *Technometrics*, 21(3):341–349, 1979.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice Hall, 2nd edition, 2008. ISBN 0-131-22798-X.

S. Kandula, R. Chandra, and D. Katabi. What's going on? learning communication rules in edge networks. In *Proc. SIGCOMM*, 2008.

Murat Kantarcioglu, Bowei Xi, and Chris Clifton. Classifier evaluation and attribute selection against active adversaries. Technical Report 09-01, Purdue University, February 2009.

Michael Kearns and Ming Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837, 1993. ISSN 0097-5397.

Auguste Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, 9:5–83, January 1883.

Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *USENIX Security Symposium*, August 2004.

Bryan Klimt and Yiming Yang. Introducing the Enron corpus. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, July 2004.

Marius Kloft and Pavel Laskov. Online anomaly detection under adversarial impact. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

Anukool Lakhina, Mark Crovella, and Christophe Diot. Characterization of network-wide anomalies in traffic flows. In Alfio Lombardo and James F. Kurose, editors, *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC)*, pages 201–206, New York, NY, USA, October 2004a. ACM. ISBN 1-58113-821-0.

Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In Raj Yavatkar, Ellen W. Zegura, and Jennifer Rexford, editors, *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 219–230, New York, NY, USA, September 2004b. ACM. ISBN 1-58113-862-8.

Anukool Lakhina, Mark Crovella, and Christophe Diot. Detecting distributed attacks using network-wide flow traffic. In *Proceedings of the FloCon 2005 Analysis Workshop*, September 2005a.

Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2005b.

Pavel Laskov and Marius Kloft. A framework for quantitative security analysis of machine learning. In *Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence (AISec)*, pages 1–4, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-781-3.

Aleksandar Lazarevic, Levent Ertöz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In Daniel Barbará and Chandrika Kamath, editors, *Proceedings of the SIAM International Conference on Data Mining*, May 2003.

Guoying Li and Zhonglian Chen. Projection-pursuit approach to robust dispersion matrices and principal components: Primary theory and Monte Carlo. *Journal of the American Statistical Association*, 80(391):759–766, September 1985.

Xin Li, Fang Bian, Mark Crovella, Christophe Diot, Ramesh Govindan, Gianluca Iannaccone, and Anukool Lakhina. Detection and identification of network anomalies using sketch subspaces. In Jussara M. Almeida, Virgílio A. F. Almeida, and Paul Barford, editors, *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement (IMC)*, pages 147–152. ACM, October 2006. ISBN 1-59593-561-4.

Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994. ISSN 0890-5401.

Changwei Liu and Sid Stamm. Fighting unicode-obfuscated spam. In *Proceedings of the Anti-Phishing Working Groups 2nd Annual eCrime Researchers Summit*, pages 45–59, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-939-8.

László Lovász and Santosh Vempala. Hit-and-run from a corner. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing (STOC)*, pages 310–314, 2004.

László Lovász and Santosh Vempala. Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 650–659, 2003.

Daniel Lowd and Christopher Meek. Good word attacks on statistical spam filters. In *Proceedings of the 2nd Conference on Email and Anti-Spam (CEAS)*, July 2005a.

Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the 11th International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 641–647, New York, NY, USA, 2005b. ACM. ISBN 1-59593-135-X.

Matthew V. Mahoney and Philip K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 376–385, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-567-X.

Matthew V. Mahoney and Philip K. Chan. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. In Giovanni Vigna, Erland Jonsson,

and Christopher Krügel, editors, *In Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 2820 of *Lecture Notes in Computer Science*, pages 220–237. Springer, September 2003. ISBN 3-540-40878-9.

Ricardo Maronna. Principal components and orthogonal regression based on robust scales. *Technometrics*, 47(3):264–273, 2005.

Ricardo A. Maronna, Douglas R. Martin, and Victor J. Yohai. *Robust Statistics: Theory and Methods*. Probability and Statistics. John Wiley and Sons, New York, NY, USA, 2006. ISBN 0-470-01092-4.

Steven L. Martin. Learning on email behavior to detect novel worm infections. Master's thesis, University of California at Berkeley, 2005.

Tony A. Meyer and Brendon Whateley. SpamBayes: Effective open-source, Bayesian based, email classification system. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, July 2004.

Tom Mitchell. *Machine Learning*. McGraw Hill, 1997. ISBN 0-07-042807-7.

Tom M. Mitchell. The discipline of machine learning. Technical Report CMU-ML-06-108, Carnegie Mellon University, 2006.

David Moore, Colleen Shannon, Douglas J. Brown, Geoffrey M. Voelker, and Stefan Savage. Inferring internet denial-of-service activity. *ACM Transactions on Computer Systems (TOCS)*, 24(2):115–139, 2006. ISSN 0734-2071.

Srinivas Mukkamala, Guadalupe Janoski, and Andrew Sung. Intrusion detection using neural networks and support vector machines. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 1702–1707, 2002.

Darren Mutz, Fredrik Valeur, Giovanni Vigna, and Christopher Kruegel. Anomalous system call detection. *ACM Transactions on Information and System Security (TISSEC)*, 9(1): 61–93, 2006. ISSN 1094-9224.

Blaine Nelson. Designing, Implementing, and Analyzing a System for Virus Detection. Master's thesis, University of California at Berkeley, December 2005.

Blaine Nelson, Deborah Schofield, and Leslie M. Collins. A comparison of neural networks and subspace detectors for the discrimination of low-metal-content landmines. In Russell S. Harmon, John H. Holloway Jr., and J. T. Broach, editors, *Proceedings of the Conference on Detection and Remediation Technologies for Mines and Minelike Targets VIII*, volume 5089, pages 1046–1053. SPIE, 2003.

Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. In *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, pages 1–9, Berkeley, CA, USA, 2008. USENIX Association.

Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, and Kai Xia. Misleading learners:

Co-opting your spam filter. In Jeffrey J. P. Tsai and Philip S. Yu, editors, *Machine Learning in Cyber Trust: Security, Privacy, Reliability*, pages 17–51. Springer, 2009.

Blaine Nelson, Benjamin I. P. Rubinstein, Ling Huang, Anthony D. Joseph, Shing hon Lau, Steven Lee, Satish Rao, Anthony Tran, and J. D. Tygar. Near-optimal evasion of convex-inducing classifiers. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010a.

Blaine Nelson, Benjamin I. P. Rubinstein, Ling Huang, Anthony D. Joseph, Steven Lee, Satish Rao, and J. D. Tygar. Query strategies for evading convex-inducing classifiers. Technical Report arXiv:1007.0484v1 [cs.LG], arXiv, July 3 2010b.

Blaine Nelson, Benjamin I. P. Rubinstein, Ling Huang, Anthony D. Joseph, and J. D. Tygar. Classifier evasion: Models and open problems (position paper). In *Proceedings of ECML/PKDD Workshop on Privacy and Security issues in Data Mining and Machine Learning (PSDML)*, September 2010c.

James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 226–241, Washington, DC, USA, May 2005. IEEE Computer Society. ISBN 0-7695-2339-0.

James Newsome, Brad Karp, and Dawn Song. Paragraph: Thwarting signature learning by training maliciously. In Diego Zamboni and Christopher Krügel, editors, *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 4219 of *Lecture Notes in Computer Science*, pages 81–105. Springer, September 2006. ISBN 3-540-39723-X.

Vern Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23):2435–2463, December 1999. ISSN 1389-1286.

Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

Réjean Plamondon and Sargur N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, January 2000. ISSN 0162-8828.

Luis Rademacher and Navin Goyal. Learning convex bodies is hard. In *Proceedings of the 22nd Annual Conference on Learning Theory (COLT)*, pages 303–308, 2009.

Anirudh Ramachandran, Nick Feamster, and Santosh Vempala. Filtering spam with behavioral blacklisting. In *Proceedings of the 14th ACM conference on Computer and communications security (CCS)*, pages 342–351, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-703-2.

Haakon Ringberg, Augustin Soule, Jennifer Rexford, and Christophe Diot. Sensitivity of PCA for traffic anomaly detection. In Leana Golubchik, Mostafa H. Ammar, and Mor Harchol-Balter, editors, *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 109–120, New York, NY, USA, June 2007. ACM. ISBN 978-1-59593-639-4.

Gary Robinson. A statistical approach to the spam problem. *Linux Journal*, March 2003.

Benjamin I. P. Rubinstein. *Secure Learning and Learning for Security: Research in the Intersection*. PhD thesis, University of California at Berkeley, May 2010.

Benjamin I. P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing hon Lau, Nina Taft, and J. D. Tygar. Compromising PCA-based anomaly detectors for network-wide traffic. Technical Report UCB/EECS-2008-73, EECS Department, University of California, Berkeley, May 2008.

Benjamin I. P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing hon Lau, Satish Rao, Nina Taft, and J. D. Tygar. ANTIDOTE: Understanding and defending against poisoning of anomaly detectors. In Anja Feldmann and Laurent Mathy, editors, *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement (IMC)*, pages 1–14, New York, NY, USA, November 2009a. ACM. ISBN 978-1-60558-771-4.

Benjamin I. P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing hon Lau, Satish Rao, Nina Taft, and J. D. Tygar. Stealthy poisoning attacks on PCA-based anomaly detectors. *SIGMETRICS Performance Evaluation Review*, 37(2):73–74, 2009b. ISSN 0163-5999.

Udam Saini. Machine learning in the presence of an adversary: Attacking and defending the spambayes spam filter. Master's thesis, University of California at Berkeley, May 2008.

Greg Schohn and David Cohn. Less is more: Active learning with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pages 839–846, 2000.

David Sculley, Gabriel M. Wachman, and Carla E. Brodley. Spam filtering using inexact string matching in explicit feature space with on-line linear classifiers. In Ellen M. Voorhees and Lori P. Buckland, editors, *Proceedings of the 15th Text REtrieval Conference (TREC)*, volume Special Publication 500-272. National Institute of Standards and Technology (NIST), November 2006.

Richard Segal, Jason Crawford, Jeff Kephart, and Barry Leiba. SpamGuru: An enterprise anti-spam filtering system. In *Conference on Email and Anti-Spam (CEAS)*, 2004.

Anil A. Sewani. A system for novel email virus and worm detection. Master's thesis, University of California at Berkeley, 2005.

Claude E. Shannon. Probability of error for optimal codes in a gaussian channel. *Bell System Technical Journal*, 38(3):611–656, May 1959.

Cyrus Shaoul and Chris Westbury. A USENET corpus (2005-2007), October 2007.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004. ISBN 0-521-81397-2.

Robert L. Smith. The hit-and-run sampler: A globally reaching Markov chain sampler for generating arbitrary multivariate distributions. In *Proceedings of the 28th Conference on Winter Simulation (WSC)*, pages 260–264, 1996.

Anil Somayaji and Stephanie Forrest. Automated response using system-call delays. In *Proceedings of the Conference on USENIX Security Symposium (SSYM)*, pages 185–197, Berkeley, CA, USA, 2000. USENIX Association.

Augustin Soule, Kavé Salamatian, and Nina Taft. Combining filtering and statistical methods for anomaly detection. In *Proceedings of the 5th Conference on Internet Measurement (IMC)*, pages 331–344. USENIX Association, October 2005.

Salvatore J. Stolfo, Shlomo Hershkop, Ke Wang, Olivier Nimeskern, and Chia-Wei Hu. A behavior-based approach to securing email systems. In *Mathematical Methods, Models and Architectures for Computer Networks Security*. Springer-Verlag, 2003.

Salvatore J. Stolfo, Wei jen Li, Shlomo Hershkop, Ke Wang, Chia wei Hu, and Olivier Nimeskern. Detecting viral propagations using email behavior profiles. In *ACM Transactions on Internet Technology (TOIT)*, May 2004.

Kymie M. C. Tan, Kevin S. Killourhy, and Roy A. Maxion. Undermining an anomaly-based intrusion detection system using common exploits. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 2516 of *Lecture Notes in Computer Science*, pages 54–73. Springer, October 2002. ISBN 3-540-00020-8.

Kymie M. C. Tan, John McHugh, and Kevin S. Killourhy. Hiding intrusions: From the abnormal to the normal and beyond. In *Revised Papers from the 5th International Workshop on Information Hiding (IH)*, pages 1–17, London, UK, 2003. Springer-Verlag. ISBN 3-540-00421-1.

John W. Tukey. A survey of sampling from contaminated distributions. *Contributions to Probability and Statistics*, pages 448–485, 1960.

Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984. ISSN 0001-0782.

Leslie G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 560–566, San Francisco, CA, USA, 1985. Morgan Kaufmann Publishers Inc. ISBN 0-934613-02-8.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995. ISBN 0-387-94559-8.

Shobha Venkataraman, Avrim Blum, and Dawn Song. Limits of learning-based signature generation with adversaries. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. The Internet Society, February 2008.

David Wagner. Resilient aggregation in sensor networks. In *Proceedings of the Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, pages 78–87, New York, NY, USA, 2004. ACM. ISBN 1-58113-972-1.

David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS)*, pages 255–264, New York, NY, USA, 2002. ACM. ISBN 1-58113-612-9.

Ke Wang, Janak J. Parekh, and Salvatore J. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In Diego Zamboni and Christopher Krügel, editors, *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 4219 of *Lecture Notes in Computer Science*, pages 226–248. Springer, September 2006. ISBN 3-540-39723-X.

Zhe Wang, William K. Josephson, Qin Lv, Moses Charikar, and Kai Li. Filtering image spam with near-duplicate detection. In *Proceedings of the 4th Conference on Email and Anti-Spam (CEAS)*, August 2007.

Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 133–145, Los Alamitos, CA, USA, 1999. IEEE Computer Society.

Matthew M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC)*, pages 61–68, Washington DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1828-1.

Gregory L. Wittel and Shyhtsun Felix Wu. On attacking statistical spam filters. In *Proceedings of the 1st Conference on Email and Anti-Spam (CEAS)*, July 2004.

Aaron D. Wyner. Capabilities of bounded discrepancy decoding. *Bell System Technical Journal*, 44:1061–1122, July/August 1965.

Wei Xu, Peter Bodík, and David A. Patterson. A flexible architecture for statistical learning and data mining from system log streams. In *Proceedings of Workshop on Temporal Data Mining: Algorithms, Theory and Applications at The 4th IEEE International Conference on Data Mining (ICDM)*, Brighton, UK, November 2004.

Yin Zhang, Zihui Ge, Albert Greenberg, and Matthew Roughan. Network anomography. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement (IMC)*, pages 317–330, Berkeley, CA, USA, October 2005. USENIX Association.

Wen-Yi Zhao, Rama Chellappa, P. Jonathon Phillips, and Azriel Rosenfeld. Face recognition: A literature survey. *ACM Computing Surveys*, 35(4):399–458, 2003.

Part III

Appendices

Appendix A

Background

A.1 Covering Hyperspheres

Here I summarize the properties of hyperspheres and spherical caps and a covering number result provided by Wyner [1965], Shannon [1959]. This covering result will be used to bound the number of queries required by any algorithm for ℓ_2 costs in Appendix C.4.

A D-dimensional *hypersphere* is simply the set of all points with ℓ_2 distance less than or equal to its radius R for its centroid (here: $\mathbf{x}^A$); *i.e.*; the ball $\mathbb{B}^R(A_2)$. Any D-dimensional *hypersphere* of radius R ,$\mathbb{S}^R$, has volume

$$vol\left(\mathbb{S}^R\right) = \frac{\pi^{\frac{D}{2}}}{\Gamma\left(1+\frac{D}{2}\right)} \cdot R^D \tag{A.1}$$

and surface area

$$surf\left(\mathbb{S}^R\right) = \frac{D \cdot \pi^{\frac{D}{2}}}{\Gamma\left(1+\frac{D}{2}\right)} \cdot R^{D-1} \ .$$

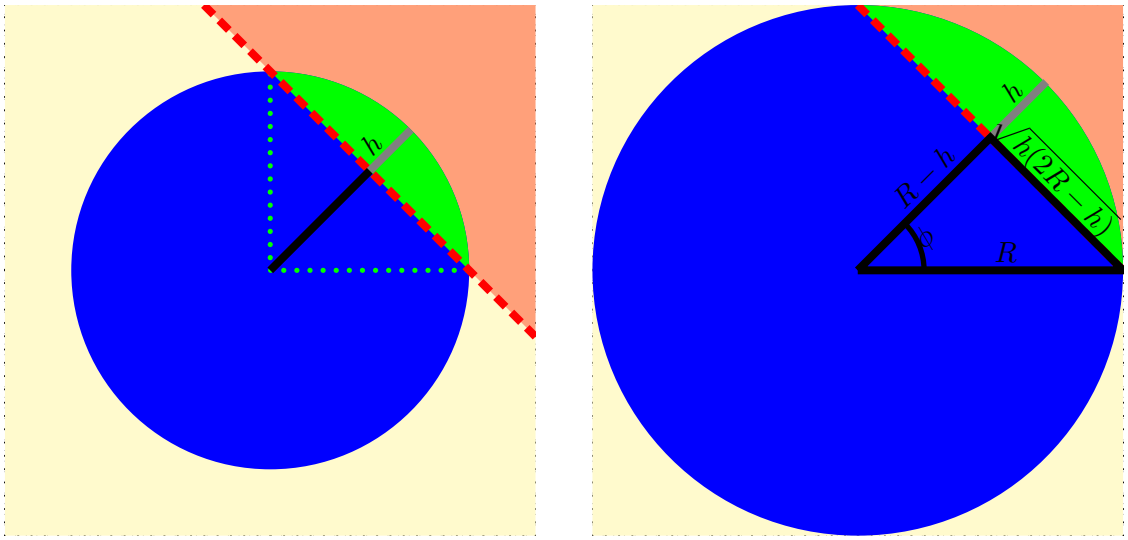
A D-dimensional *spherical cap* is the region formed by the intersection of a halfspace and a hypersphere facing away from the center of the hypersphere as depicted in Figure A.1(a). The cap has a height of h which represents the maximum length between the plane and the spherical arc. A cap of height h on a D-dimensional hypersphere of radius R will be denoted by $\mathbb{C}_h^R$ and has a volume

$$vol\left(\mathbb{C}_h^R\right) = \frac{\pi^{\frac{D-1}{2}} R^D}{\Gamma\left(\frac{D+1}{2}\right)} \int_0^{\arccos\left(\frac{R-h}{R}\right)} \sin^D(t)\,\mathrm{d}t$$

and a surface area

$$surf\left(\mathbb{C}_h^R\right) = \frac{(D-1)\cdot\pi^{\frac{D-1}{2}} R^{D-1}}{\Gamma\left(\frac{D+1}{2}\right)} \int_0^{\arccos\left(\frac{R-h}{R}\right)} \sin^{D-2}(t)\,\mathrm{d}t \ .$$

Alternatively, the cap can be parameterized in terms of the hypersphere's radius R and the half-angle ϕ about a central radius (through the peak of the cap) as in Figure A.1(b). A cap of half angle ϕ forms the right triangle depicted in the figure, for which $R - h = R\cos(\phi)$



(a) A Spherical Cap on a Circle

(b) An Angular Cap on a Circle

Figure A.1: This figure shows various depictions of spherical caps. **(a)** A depiction of a spherical cap of height h that is created by a halfspace that passes through the sphere. The green region represents the area of the cap. **(b)** The geometry of the spherical cap; the intersecting halfspace forms a right triangle with the centroid of the hypersphere. The length of the side of this triangle adjacent to the centroid is $R-h$, its hypotenuse has length R, and the side opposite the centroid has length $\sqrt{h(2R-h)}$. The half angle ϕ given by $\sin(\phi) = \frac{\sqrt{h(2R-h)}}{R}$ of the right circular cone can also be used to parameterize the cap.

so that h can be expressed in terms of R and ϕ as $h = R * (1 - \cos\phi)$. Substituting this expression for h into the above formulas yields the volume of the cap as

$$vol\left(\mathbb{C}_\phi^R\right) = \frac{\pi^{\frac{D-1}{2}} R^D}{\Gamma\left(\frac{D+1}{2}\right)} \int_0^\phi \sin^D(t)\,\mathrm{d}t \tag{A.2}$$

and its surface area as

$$surf\left(\mathbb{C}_\phi^R\right) = \frac{(D-1)\cdot\pi^{\frac{D-1}{2}} R^{D-1}}{\Gamma\left(\frac{D+1}{2}\right)} \int_0^\phi \sin^{D-2}(t)\,\mathrm{d}t \ .$$

Based on these formulas, I now bound the number of spherical caps of half-angle ϕ required to cover the sphere mirroring the result in Wyner [1965]; *Capabilities of Bounded Discrepancy Decoding.*

Lemma A.1. ***(Result based on Wyner [1965])*** *Covering the surface of D-dimensional hypersphere of radius R, $\mathbb{S}^R$, requires at least*

$$\left(\frac{1}{\sin(\phi)}\right)^{D-2}$$

spherical caps of half-angle ϕ.

Proof. Suppose there are M caps that cover the hypersphere. The total surface area of the M caps must be at least the surface area of the hypersphere. Thus,

$$\begin{aligned}
M &\geq \frac{surf\left(\mathbb{S}^R\right)}{surf\left(\mathbb{C}_\phi^R\right)} \\
&\geq \frac{\frac{D\cdot\pi^{\frac{D}{2}}}{\Gamma\left(1+\frac{D}{2}\right)}\cdot R^{D-1}}{\frac{(D-1)\cdot\pi^{\frac{D-1}{2}}R^{D-1}}{\Gamma\left(\frac{D+1}{2}\right)}\int_0^\phi \sin^{D-2}(t)\,\mathrm{d}t} \\
&\geq \frac{D\sqrt{\pi}\Gamma\left(\frac{D+1}{2}\right)}{(D-1)\Gamma\left(1+\frac{D}{2}\right)}\left[\int_0^\phi \sin^{D-2}(t)\,\mathrm{d}t\right]^{-1} \ ,
\end{aligned}$$

which is the result derived by Wyner (although applied as a bound on the packing number rather than the covering number). I continue by lower bounding the above integral. As demonstrated above, integrals of the form $\int_0^\phi \sin^D(t)\,\mathrm{d}t$ arise in computing the volume or surface area of a spherical cap. To upper bound the volume of such a cap, note that *i*) the spherical cap is defined by a hypersphere and a hyperplane, *ii*) their intersection form a $(D-1)$-dimensional hypersphere as the base of the cap, *iii*) the projection of the center of the first hypersphere onto the hyperplane is the center of the $(D-1)$-dimensional hyperspherical intersection, *iv*) the distance between these centers is $R - h$, and *v*) this projected point achieves the maximum height of the cap; *i.e.*, continuing along the radial line achieves the remaining distance h—the height of the cap. I use these facts to upper bound the volume of the cap by enclosing the cap within a D-dimensional hypersphere. As seen in Figure A.1(b), the center of the $(D-1)$-dimensional hyperspherical intersection forms a right triangle with the original hypersphere's center and the edge of the intersecting

spherical region (by symmetry, all such edge points are equivalent). That right triangle has one side of length $R - h$ and a hypotenuse of R. Hence, the other side has length $s = \sqrt{h(2R-h)} = R\sin(\phi)$. Moreover, $R \geq h$ implies $s \geq h$. Thus, a D-dimensional hypersphere of radius s encloses the cap and its volume from Equation (A.1) bounds the volume of the cap as

$$vol\left(\mathbb{C}_{\phi}^{R}\right) \leq vol\left(\mathbb{S}^{s}\right) = \frac{\pi^{\frac{D}{2}}}{\Gamma\left(1+\frac{D}{2}\right)} \cdot \left(R\sin(\phi)\right)^{D} \quad .$$

Applying this bound to the formula for the volume of the cap in Equation A.2 then yields the following bound on the integral:

$$\begin{aligned} \frac{\pi^{\frac{D-1}{2}} R^{D}}{\Gamma\left(\frac{D+1}{2}\right)} \int_{0}^{\phi} \sin^{D}(t)\,\mathrm{d}t &\leq \frac{\pi^{\frac{D}{2}}}{\Gamma\left(1+\frac{D}{2}\right)} \cdot \left(R\sin(\phi)\right)^{D} \\ \int_{0}^{\phi} \sin^{D}(t)\,\mathrm{d}t &\leq \frac{\sqrt{\pi}\Gamma\left(\frac{D+1}{2}\right)}{\Gamma\left(1+\frac{D}{2}\right)} \cdot \sin^{D}(\phi) \quad . \end{aligned}$$

Using this bound on the integral, the bound on the size of the covering from Wyner reduces to the following (weaker) bound

$$M \geq \frac{D\sqrt{\pi}\Gamma\left(\frac{D+1}{2}\right)}{(D-1)\Gamma\left(1+\frac{D}{2}\right)} \left[\frac{\sqrt{\pi}\Gamma\left(\frac{D-1}{2}\right)}{\Gamma\left(\frac{D}{2}\right)} \cdot \sin^{D-2}(\phi)\right]^{-1} \quad .$$

Finally, using properties of the gamma function, it can be shown that $\frac{\Gamma\left(\frac{D+1}{2}\right)\Gamma\left(\frac{D}{2}\right)}{\Gamma\left(1+\frac{D}{2}\right)\Gamma\left(\frac{D-1}{2}\right)} = \frac{D-1}{D}$ which simplifies the above expression to

$$M \geq \left(\frac{1}{\sin(\phi)}\right)^{D-2} \quad .$$

□

It is worth noting that by further bounding the integral $\int_{0}^{\phi} \sin^{D}(t)\,\mathrm{d}t$, the bound in Lemma A.1 is weaker than the original bound on the covering derived in Wyner [1965]. However, the bound provided by the lemma is more useful for later results because it is expressed in a closed form (see the proof for Theorem 6.10 in Appendix C.4).

Of course, there are other tighter bounds on the power-of-sine integral. In Lemma A.1, this quantity was bounded using a bound on the volume of a spherical cap, but here I instead bound the integral directly. A naive bound can be accomplished by observing that all the terms in the integral are less than the final term, which yields

$$\int_{0}^{\phi} \sin^{D}(t)\,\mathrm{d}t \leq \phi \cdot \sin^{D}(\phi) \quad ,$$

but this bound is looser than the bound achieved in the lemma. However, by first performing a variable substitution, a tighter bound on the integral can be obtained. The variable substitution is given by letting $p = \sin^{2}(t)$, $t = \arcsin\left(\sqrt{p}\right)$, and $\mathrm{d}t = \frac{\mathrm{d}p}{2\sqrt{1-p}\sqrt{p}}$. This yields

$$\int_{0}^{\phi} \sin^{D}(t)\,\mathrm{d}t = \frac{1}{2}\int_{0}^{\sin^{2}(\phi)} \frac{p^{\frac{D-1}{2}}}{\sqrt{1-p}}\mathrm{d}p \quad .$$

Within the integral, the denominator is monotonically decreasing in p since, for the interval of integration, $p \leq 1$. Thus it achieves its minimum value at the upper limit $p = \sin^2(\phi)$. Fixing the denominator at this value therefore results in the following upper bound on the integral:

$$\int_0^{\phi} \sin^D(t)\,\mathrm{d}t \leq \frac{1}{2\cos(\phi)} \int_0^{\sin^2(\phi)} p^{\frac{D-1}{2}}\,\mathrm{d}p = \frac{\sin^{D+1}(\phi)}{(D+1)\cos(\phi)} \ . \tag{A.3}$$

This bound is not strictly tighter than the bound applied in Lemma A.1, but for large D and $\phi < \frac{\pi}{2}$, this result does achieve a tighter bound. I apply this bound for additional analysis in Chapter 6.3.1.4.

A.2 Covering Hypercubes

Here I introduce results for covering a D-dimensional *hypercube graphs*—a collection of 2^D nodes of the form $\langle \pm 1, \pm 1, \ldots, \pm 1 \rangle$ where each node has an edge to every other node that is Hamming distance 1 from it. The following lemma summarizes coverings of a hypersphere and is utilized in Appendix C.3 for a general query complexity result for ℓ_p distances:

Lemma A.2. *For any $0 < \delta < \frac{1}{2}$, to cover a D-dimensional hypercube graph so that every vertex has a Hamming distance of at most $\lfloor \delta D \rfloor$ to some vertex in the covering, the number of vertices in the covering must be*

$$Q(D, h) \geq 2^{D(1-H(\delta))} \ ,$$

where $H(\delta) = -\delta \log_2(\delta) - (1-\delta)\log_2(1-\delta)$ is the entropy *of δ.*

Proof. There are 2^D vertices in the D-dimensional hypercube graph. Each vertex in the covering is within a Hamming distance of at most h for exactly $\sum_{k=0}^{h} \binom{D}{k}$ vertices. Thus, one needs at least $2^D / \left(\sum_{k=0}^{h} \binom{D}{k}\right)$ to cover the hypercube graph. Now I apply the bound

$$\sum_{k=0}^{\lfloor \delta D \rfloor} \binom{D}{k} \leq 2^{H(\delta)D}$$

to the denominator, which is valid for any $0 < \delta < \frac{1}{2}$. □

Lemma A.3. *The minimizer of the ℓ_p cost function A_p to any target $\mathbf{x}^A$ on the halfspace $\mathbb{H}^{(\mathbf{w},\mathbf{b})} = \left\{\mathbf{x} \;\middle|\; \mathbf{x}^\top \mathbf{w} \geq \mathbf{b}^\top \mathbf{w}\right\}$ can be expressed in terms of the equivalent hyperplane $\mathbf{x}^\top \mathbf{w} \geq d$ parameterized by a normal vector $\mathbf{w}$ and displacement $d = \left(\mathbf{b} - \mathbf{x}^A\right)^\top \mathbf{w}$ as*

$$\begin{cases} d \cdot \|\mathbf{w}\|_{\frac{p}{p-1}}^{-1} \ , & \text{if } d > 0 \\ 0 \ , & \text{otherwise} \end{cases} \tag{A.4}$$

for all $1 < p < \infty$ and is

$$\begin{cases} d \cdot \|\mathbf{w}\|_1^{-1} \ , & \text{if } d > 0 \\ 0 \ , & \text{otherwise} \end{cases} \tag{A.5}$$

for $p = \infty$.

Proof. For $1 < p < \infty$, minimizing A_p on the halfspace $\mathbb{H}^{(\mathbf{w},\mathbf{b})}$ is equivalent to finding a minimizer for

$$\min_{\mathbf{x}} \frac{1}{p} \sum_{i=1}^{D} |x_i|^p \quad \text{s.t.} \quad \mathbf{x}^\top \mathbf{w} \leq d \ .$$

Clearly, if $d \leq 0$ then the vector $\mathbf{0}$ (corresponding to $\mathbf{x}^A$ in the transformed space) trivially satisfies the constraint and minimizes the cost function with cost 0 which yields the second case of Equation (A.4). For the case $d > 0$, I construct the Lagrangian

$$\mathcal{L}(\mathbf{x}, \lambda) \triangleq \frac{1}{p} \sum_{i=1}^{D} |x_i|^p - \lambda \left(\mathbf{x}^\top \mathbf{w} - d \right) \ .$$

Differentiating this with respect to $\mathbf{x}$ and setting that partial derivative equal to zero yields

$$x_i^* = \operatorname{sign}(w_i) (\lambda |w_i|)^{\frac{1}{p-1}} \ .$$

Plugging this back into the Lagrangian yields

$$\mathcal{L}(\mathbf{x}^*, \lambda) = \frac{1-p}{p} \lambda^{\frac{p}{p-1}} \sum_{i=1}^{D} |w_i|^{\frac{p}{p-1}} + \lambda d \ ,$$

which I differentiate with respect to λ and set the derivative equal to zero to yield

$$\lambda^* = \left(\frac{d}{\sum_{i=1}^{D} |w_i|^{\frac{p}{p-1}}} \right)^{p-1} \ .$$

Plugging this solution into the formula for $\mathbf{x}^*$ yields the solution

$$x_i^* = \operatorname{sign}(w_i) \left(\frac{d}{\sum_{i=1}^{D} |w_i|^{\frac{p}{p-1}}} \right) |w_i|^{\frac{1}{p-1}} \ .$$

The ℓ_p cost of this optimal solution is given by

$$A_p(\mathbf{x}^*) = d \cdot \|\mathbf{w}\|_{\frac{p}{p-1}}^{-1} \ ,$$

which is the first case of Equation (A.4).

For $p = \infty$, once again if $d \leq 0$ then the vector $\mathbf{0}$ trivially satisfies the constraint and minimizes the cost function with cost 0 which yields the second case of Equation (A.5). For the case $d > 0$, I use the geometry of hypercubes (the equi-cost balls of a ℓ_∞ cost function) to derive the second case of Equation (A.5). Any optimal solution must occur at a point where the hyperplane given by $\mathbf{x}^\top \mathbf{w} = \mathbf{b}^\top \mathbf{w}$ is tangent to a hypercube about $\mathbf{x}^A$—this can either occur along a side (face) of the hypercube or at a corner. However, if the plane is tangent along a side (face) it is also tangent at a corner of the hypercube. Hence, there is always an optimal solution at some corner of optimal cost hypercube.

The corner of the hypercube has the following property:

$$|x_1^*| = |x_2^*| = \ldots = |x_D^*| \ ;$$

that is, the magnitude of all coordinates of this optimal solution is the same value. Further, the sign of the optimal solution's i^{th} coordinate must agree with the sign of the hyperplane's i^{th} coordinate, w_i. These constraints, along with the hyperplane constraint, lead to the following formula for an optimal solution:

$$x_i = d \cdot \text{sign}\left(w_i\right) \|\mathbf{w}\|_1^{-1} \quad .$$

The ℓ_∞ cost of these solutions is simply $d \cdot \|\mathbf{w}\|_1^{-1}$. □

Appendix B

Analysis of SpamBayes

In this appendix, I analyze the effect of attack messages on SpamBayes. This analysis serves as the motivation for the attacks presented in Chapter 4.3.

B.1 SpamBayes' $I(\cdot)$ Message Score

As mentioned in Chapter 4.1.1, the SpamBayes $I(\cdot)$ function used to estimate spaminess of a message, is the average between its score $S(\cdot)$ and one minus its score $H(\cdot)$. Both of these scores are expressed in terms of the *chi-squared* cumulative distribution function (CDF): $\chi^2_{2n}(\cdot)$. In both these score functions, the argument to the CDF is an inner product between the logarithm of a scores vector and the indicator vector $\boldsymbol{\delta}(\hat{\mathbf{x}})$ as in Equation (4.3). These terms can be re-arranged to rewrite these functions as $S(\hat{\mathbf{x}}) = 1 - \chi^2_{2n}(-2\log s_{\mathbf{q}}(\hat{\mathbf{x}}))$ and $H(\hat{\mathbf{x}}) = 1 - \chi^2_{2n}(-2\log h_{\mathbf{q}}(\hat{\mathbf{x}}))$ where $s_{\mathbf{q}}(\cdot)$ and $h_{\mathbf{q}}(\cdot)$ are scalar functions that map $\hat{\mathbf{x}}$ onto $[0,1]$ defined as

$$s_{\mathbf{q}}(\hat{\mathbf{x}}) \triangleq \prod_i q_i^{\delta(\hat{\mathbf{x}})_i} \tag{B.1}$$

$$h_{\mathbf{q}}(\hat{\mathbf{x}}) \triangleq \prod_i (1-q_i)^{\delta(\hat{\mathbf{x}})_i} \ . \tag{B.2}$$

I further explore these functions in the next section, but first I expound on the properties of $\chi^2_k(\cdot)$.

The $\chi^2_k(\cdot)$ CDF can be written out exactly using gamma functions. For $k \in \aleph$ and $x \in \Re^{0+}$ it is simply

$$\chi^2_k(x) = \frac{\gamma(k/2, x/2)}{\Gamma(k/2)}$$

where the *lower-incomplete gamma function* is $\gamma(k,y) = \int_0^y t^{k-1}e^{-t}dt$, the *upper-incomplete gamma function* is $\Gamma(k,y) = \int_y^\infty t^{k-1}e^{-t}dt$, and the *gamma function* is $\Gamma(k) = \int_0^\infty t^{k-1}e^{-t}dt$. By these definitions, it follows that for any k and y, the gamma functions are related by $\Gamma(k) = \gamma(k,x) + \Gamma(k,x)$. Also note that for $k \in \aleph$

$$\Gamma(k,y) = (k-1)!\, e^{-y} \sum_{j=0}^{k-1} \frac{y^j}{j!} \qquad \Gamma(k) = (k-1)! \ .$$

Based on these properties, the $S(\cdot)$ score can be rewritten as

$$\begin{aligned} S(\hat{\mathbf{x}}) &= \frac{\Gamma(n, -\log s_{\mathbf{q}}(\hat{\mathbf{x}}))}{\Gamma(n)} = s_{\mathbf{q}}(\hat{\mathbf{x}}) \sum_{j=0}^{n-1} \frac{(-\log s_{\mathbf{q}}(\hat{\mathbf{x}}))^j}{j!} \\ H(\hat{\mathbf{x}}) &= \frac{\Gamma(n, -\log h_{\mathbf{q}}(\hat{\mathbf{x}}))}{\Gamma(n)} = h_{\mathbf{q}}(\hat{\mathbf{x}}) \sum_{j=0}^{n-1} \frac{(-\log h_{\mathbf{q}}(\hat{\mathbf{x}}))^j}{j!} \quad . \end{aligned}$$

It is easy shown that both these functions are monotonically non-decreasing in $s_{\mathbf{q}}(\hat{\mathbf{x}})$ and $h_{\mathbf{q}}(\hat{\mathbf{x}})$ respectively. For either of these functions, the following derivative can be taken (with respect to $s_{\mathbf{q}}(\hat{\mathbf{x}})$ or $h_{\mathbf{q}}(\hat{\mathbf{x}})$):

$$\frac{d}{dx}\left[x \sum_{j=0}^{n-1} \frac{(-\log x)^j}{j!}\right] = \frac{1}{(n-1)!}(-\log x)^{n-1} \quad ,$$

which is non-negative for $0 \leq x \leq 1$.

B.2 Constructing Optimal Attacks on SpamBayes

As indicated by Equation (4.7) in Chapter 4.3.1, an attacker with objectives described in Chapter 4.2.1 would like to have the maximal (deleterious) impact on the performance of SpamBayes. In this section, I analyze SpamBayes' decision function $I(\cdot)$ to optimize the attacks' impact. Here I show that the attacks proposed in Chapter 4.3.1 are (nearly) optimal strategies for designing a single attack message that maximally increases $I(\cdot)$.

In the attack scenario described in Chapter 4.3.1.1, the attacker will send a series of attack messages which will increase $N^{(s)}$ and $n_j^{(s)}$ for the tokens that are included in the attacks. I will show how $I(\cdot)$ changes as the token counts $n_j^{(s)}$ are increased to understand which tokens the attacker should choose to maximize the impact per message. This analysis separates into two parts based on the following observation.

Remark Given a fixed number of attack spam messages, q_j is independent of the number of those messages containing the k^{th} token for all $k \neq j$.

This remark follows from the fact that the inclusion of the j^{th} token in attack spams affects $n_j^{(s)}$ and n_j but not $n_k^{(h)}$, $N^{(s)}$, $N^{(h)}$, $n_k^{(s)}$, $n_k^{(h)}$, or n_k for all $k \neq j$ (see Equations (4.1) and (4.2) in Chapter 4.1.1).

After an attack consisting of a fixed number of attack spam messages, the score $I(\hat{\mathbf{x}})$ of an incoming test message $\hat{\mathbf{x}}$ can be maximized by maximizing each q_j separately. This motivates dictionary attacks and focused attacks—intuitively, the attacker would like to maximally increase the q_j of tokens appearing (or most likely to appear) in $\hat{\mathbf{x}}$ depending on the information the attacker has about future messages.

Thus, I first analyze the effect of increasing $n_j^{(s)}$ on its score q_i in Section B.2.1. Based on this, I subsequently analyze the change in $I(\hat{\mathbf{x}})$ that is caused altering the token score q_i in Section B.2.2. As one might expect, since increasing the number of occurrences of the j^{th}

token in spam should increase the posterior probability that a message with the j^{th} token is spam, I show that including the j^{th} token in an attack message generally increases the corresponding score q_j more than not including that token (except in unusual situations which I identify below). Similarly, I show that increasing q_j generally increases the overall spam score $I(\cdot)$ of a message containing the j^{th} token. Based on these results, I motivate the attack strategies presented in Chapter 4.3.1.

B.2.1 Effect of Poisoning on Token Scores

In this section, I establish how token spam scores change as the result of attack messages in the training set. Intuitively, one might expect that the j^{th} score q_j should increase when the j^{th} token is added to the attack email. This would be the case, in fact, if the token score in Equation (4.1) were computed according to Bayes' Rule. However, as noted Chapter 4.1, the score in Equation (4.1) is derived by applying Bayes' Rule with an additional assumption that the prior of spam and ham is equal. As a result, there are circumstances in which the spam score q_j can decrease when the j^{th} token is included in the attack email—specifically when the assumption is violated. Here, I show that this occurs when there is an extraordinary imbalance between the number of ham and spam in the training set.

As in Chapter 4.3, I consider an attacker whose attack messages are composed a single set of attack tokens; *i.e.*, each token is either included in *all* attack messages or *none*. In this fashion, the attacker creates a set of k attack messages used in the retraining of the filter, after which the counts become

$$\begin{aligned}
N^{(s)} &\mapsto N^{(s)} + k \\
N^{(h)} &\mapsto N^{(h)} \\
n_j^{(s)} &\mapsto \begin{cases} n_j^{(s)} + k \ , & \text{if } a_j = 1 \\ n_j^{(s)} \ , & \text{otherwise} \end{cases} \\
n_j^{(h)} &\mapsto n_j^{(h)} \ .
\end{aligned}$$

Using these count transformations, I compute the difference in the smoothed SpamBayes score q_j between training on an attack spam message $\mathbf{a}$ that contains the j^{th} token and an attack spam that does not contain it. If the j^{th} token is included in the attack (*i.e.*, $a_j = 1$), then the new score for the j^{th} token (from Equation 4.1) is

$$P_j^{(s,k)} \triangleq \frac{N^{(h)}\left(n_j^{(s)} + k\right)}{N^{(h)}\left(n_j^{(s)} + k\right) + \left(N^{(s)} + k\right) n_j^{(h)}} \ .$$

If the token is not included in the attack (*i.e.*, $a_j = 0$), then the new token score is

$$P_j^{(s,0)} \triangleq \frac{N^{(h)} n_j^{(s)}}{N^{(h)} n_j^{(s)} + \left(N^{(s)} + k\right) n_j^{(h)}} \ .$$

Similarly, I use $q_j^{(k)}$ and $q_j^{(0)}$ to denote the smoothed spam score after the attack depending on whether or not the j^{th} token was used in the attack message. I will analyze the quantity

$$\Delta^{(k)} q_j \triangleq q_j^{(k)} - q_j^{(0)} \ .$$

One might reasonably expect this difference to always be non-negative, but here I show that there are some scenarios in which $\Delta^{(k)} q_j < 0$. This unusual behavior is a direct result of the assumption made by SpamBayes that $N^{(h)} = N^{(s)}$ rather than using a proper prior distribution. In fact, it can be shown that the usual spam model depicted in Figure 4.1(b) does not exhibit these irregularities. Below, I will show how SpamBayes' assumption can lead to situations where $\Delta^{(k)} q_j < 0$ but also that these irregularities only occur when there is *many* more spam messages than ham messages in the training dataset. By expanding $\Delta^{(k)} q_j$ and rearranging terms, the difference can be expressed as:

$$\begin{aligned} \Delta^{(k)} q_j &= \frac{s \cdot k}{(s + n_j + k)(s + n_j)} \left(P_j^{(s,k)} - x \right) \\ &\quad + \frac{k \cdot N^{(h)} \cdot n_j}{(s + n_j)\left(N^{(h)} \cdot n_j^{(s)} + \left(N^{(s)} + k \right) n_j^{(h)} \right)} P_j^{(h,k)} \quad , \end{aligned}$$

where $P_j^{(h,k)} = 1 - P_j^{(s,k)}$ is the altered ham score of the j^{th} token. The difference can be rewritten as

$$\begin{aligned} \Delta^{(k)} q_j &= \frac{k}{(s + n_j + k)(s + n_j)} \cdot \alpha_j \\ \alpha_j &\triangleq s(1 - x) \\ &\quad + P_j^{(h,k)} \cdot \frac{N^{(h)} \cdot n_j (n_j + k) + s \cdot N^{(h)} \cdot n_j^{(h)} - s \left(N^{(s)} + k \right) n_j^{(h)}}{N^{(h)} \cdot n_j^{(s)} + \left(N^{(s)} + k \right) n_j^{(h)}} \quad . \end{aligned}$$

The first factor $\frac{k}{(s+n_j+k)(s+n_j)}$ in the above expression is non-negative so only α_j can make $\Delta^{(k)} q_j$ negative. From this, it is easy to show that $N^{(s)} + k$ must be greater that $N^{(h)}$ for $\Delta^{(k)} q_j$ to be negative, but I demonstrate stronger conditions. Generally, I demonstrate that for $\Delta^{(k)} q_j$ to be negative there must be a large disparity between the number of spams after the attack, $N^{(s)} + numAttacks$, and the number of hams, $N^{(h)}$. This reflects the effect of violating the implicit assumption made by SpamBayes that $N^{(h)} = N^{(s)}$.

Expanding the expression for α_j, the following condition is necessary for $\Delta^{(k)} q_j$ to be negative:

$$\begin{aligned} \frac{s \left(N^{(s)} + k \right) n_j^{(h)} x}{N^{(h)}} &> \frac{s(1-x)\left(n_j^{(s)} + k \right)}{n_j^{(h)} \left(N^{(s)} + k \right)} \left[\left(N^{(s)} + k \right) n_j^{(h)} + N^{(h)} \cdot n_j^{(s)} \right] \\ &\quad + \quad n_j (n_j + k) + s n_j^{(s)} (1 - x) + s \cdot n_j^{(h)} \end{aligned} \quad .$$

Because $1 - x \geq 0$ (since $x \leq 1$) and $n_j = n_j^{(s)} + n_j^{(h)}$, the right-hand side of the above expression is strictly increasing in $n_j^{(s)}$ while the left-hand side is constant in $n_j^{(s)}$. Thus, the weakest condition to make $\Delta^{(k)} q_j$ negative occurs when $n_j^{(s)} = 0$; *i.e.*, tokens that were not observed in any spam prior to the attack are most susceptible to having $\Delta^{(k)} q_j < 0$ while tokens that were observed more frequently in spam prior to the attack require an increasingly larger disparity between $N^{(h)}$ and $N^{(s)}$ for $\Delta^{(k)} q_j < 0$ to occur. Here I analyze the case when $n_j^{(s)} = 0$ and, using the previous constraints that $s > 0$ and $n_j^{(h)} > 0$, I arrive

at the weakest condition for which $\Delta^{(k)} q_j$ can be negative. This condition can be expressed succinctly as the following condition on $x$for the attack to cause a token's score to decrease[1]:

$$x \quad > \quad \frac{N^{(h)} \left(n_j^{(h)} + s\right) \left(n_j^{(h)} + k\right)}{s \left(n_j^{(h)} \left(N^{(s)} + k\right) + k N^{(h)}\right)} \ .$$

First, notice that the right-hand side is always positive; *i.e.*, there will always be some non-trivial threshold on the value of x to allow for $\Delta^{(k)} q_j$ to be negative. Further, when the right-hand side of this bound is at least one, there are no tokens that have a negative $\Delta^{(k)} q_j$ since the parameter $x \in [0, 1]$. For instance, this occurs when $n_j^{(h)} = 0$ or when $N^{(h)} \geq N^{(s)} + k$ (as previously noted).

Reorganizing the terms, the bound on the number of spams can be expressed as,

$$N^{(s)} + k \quad > \quad N^{(h)} \cdot \frac{\left(n_j^{(h)}\right)^2 + (s + k)\, n_j^{(h)} + s\,(1 - x)\, k}{s n_j^{(h)} x} \ .$$

This bound shows that the number of spam after the attack, $N^{(s)} + k$, must be larger than a multiple of total number of ham, $N^{(h)}$, to have any token with $\Delta^{(k)} q_j < 0$. The factor in this multiple is always greater than one, but depends on the $n_j^{(h)}$ of the j^{th} token. In fact, the factor is strictly increasing in $n_j^{(h)}$; thus, the weakest bound occurs when $n_j^{(h)} = 1$ (recall that when $n_j^{(h)} = 0$, $\Delta^{(k)} q_j$ is always non-negative). When we examine SpamBayes' default values of $s = 1$ and $x = \frac{1}{2}$, the weakest bound (for tokens with $n_j^{(h)} = 1$ and $n_j^{(s)} = 0$) is

$$N^{(s)} + k > N^{(h)} \cdot (4 + 3k)$$

Thus, when the number of spam after the attack, $N^{(s)} + k$, is sufficiently larger than the number of ham, $N^{(h)}$, it is possible that the score of a token will be lower if it is *included* in the attack message than if it were excluded. This is a direct result of the assumption made by SpamBayes that $N^{(s)} = N^{(h)}$. I have shown that such aberrations will occur most readily in tokens with low initial values of $n_j^{(h)}$ and $n_j^{(s)}$; *i.e.*, those seen infrequently in the dataset. However, for any significant number of attacks, k, the disparity between $N^{(s)} + k$ and $N^{(s)}$ must be tremendous for such aberrations to occur. Under the default SpamBayes settings, there would have to be at least 7 times as many spam as ham with only a single attack message. For more attack messages ($k > 1$), this bound is even greater. Thus, in designing attacks against SpamBayes, I ignore the extreme cases outlined here and I assume that $\Delta^{(k)} q_j$ always increases if the j^{th} token is included in the attack. Further, none of the experiments presented in Chapter 4.5 meet the criteria required to have $\Delta^{(k)} q_j < 0$.

B.2.2 Effect of Poisoning on $I(\cdot)$

The key to understanding effect of attacks and constructing optimal attacks against SpamBayes is characterizing conditions under which SpamBayes' score $I(\hat{\mathbf{x}})$ increases when the

[1]In the case that $n_j^{(s)} > 0$, the condition is stronger but the expression is more complicated.

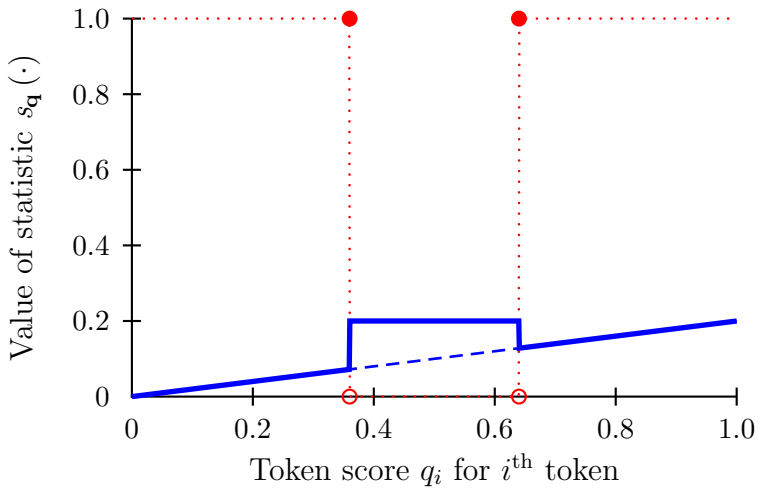


Figure B.1: Plot of the aggregation statistic $s_{\mathbf{q}}(\cdot)$ relative to a single token score q_i; on the x-axis is q_i and on the y-axis is $s_{\mathbf{q}}(\cdot)$. Here I consider a scenario where $\tau_{=}0.14$ and without the i^{th} token $s_{\mathbf{q}}(\hat{\mathbf{x}} \setminus \{i\}) = 0.2$. The red dotted line is the value of $\delta(\hat{\mathbf{x}})_i$, the blue dotted line is the value of $q_i \prod_{j \neq i} q_j$ (*i.e.*, $s_{\mathbf{q}}(\hat{\mathbf{x}})$ without including $\boldsymbol{\delta}(\hat{\mathbf{x}})$), and the blue solid line is the value of $s_{\mathbf{q}}(\hat{\mathbf{x}})$ as q_i varies.

training corpus is injected with attack spam messages. To do this, I dissect the method used by SpamBayes to aggregate token scores.

The statistics $s_{\mathbf{q}}(\hat{\mathbf{x}})$ and $h_{\mathbf{q}}(\hat{\mathbf{x}})$ from Equation (B.1) and (B.2) are measures of the *spaminess* and *haminess* of the message represented by $\hat{\mathbf{x}}$, respectively. Both assume that each token in the message presents an assessment of the *spaminess* of the message—the score q_i is the evidence for spam given by observing the i^{th} token. Further, by assuming independence, $s_{\mathbf{q}}(\hat{\mathbf{x}})$ and $h_{\mathbf{q}}(\hat{\mathbf{x}})$ aggregate this evidence into a measure of the overall message's *spaminess*. For instance, if all tokens have $q_i = 1$, $s_{\mathbf{q}}(\hat{\mathbf{x}}) = 1$ indicates that the message is very *spammy* and $1 - h_{\mathbf{q}}(\hat{\mathbf{x}}) = 1$ concurs. Similarly, when all tokens have $q_i = 0$, both scores indicate that the message is ham.

These statistics also are (almost) nicely behaved. If we instead consider the ordinary product of the scores of all tokens in the message $\hat{\mathbf{x}}$, $\tilde{s}_{\mathbf{q}}(\hat{\mathbf{x}}) \triangleq \prod_{i:\hat{x}_i=1} q_i$, it is a linear function with respect to each q_i, and is monotonically non-decreasing. Similarly, the product $\tilde{h}_{\mathbf{q}}(\hat{\mathbf{x}}) \triangleq \prod_{i:\hat{x}_i=1}(1 - q_i)$ is linear with respect to each q_i and is monotonically non-increasing. Thus, if we increase any score q_i, the first product will not decrease and the second will not increase, as expected[2]. In fact, by redefining the scores $I(\cdot)$, $S(\cdot)$, and $H(\cdot)$ in terms of the simple products $\tilde{s}_{\mathbf{q}}(\hat{\mathbf{x}})$ and $\tilde{h}_{\mathbf{q}}(\hat{\mathbf{x}})$ (which I refer to as $\tilde{I}(\cdot)$, $\tilde{S}(\cdot)$, and $\tilde{H}(\cdot)$, respectively), the following lemma shows that $\tilde{I}(\cdot)$ is non decreasing in q_i.

Lemma B.1. *The modified $\tilde{I}(\hat{\mathbf{x}})$ score is non-decreasing in q_i for all tokens (indexed by i).*

[2]These statistics do behave oddly in another sense; adding an additional token will always decrease both products and removing a token will always increase both products. Applying the chi-squared distribution rectifies this effect.

Proof. I show that the derivative of $\tilde{I}(\hat{\mathbf{x}})$ with respect to q_k is non-negative for all k. By rewriting, Equation (4.3) in terms of $\tilde{s}_{\mathbf{q}}(\hat{\mathbf{x}})$ as $\tilde{S}(\hat{\mathbf{x}}) = 1 - \chi^2_{2n}(-2\log(\tilde{s}_{\mathbf{q}}(\hat{\mathbf{x}})))$, the chain rule can be applied as follows:

$$\begin{aligned}\frac{\partial}{\partial q_k}\tilde{S}(\hat{\mathbf{x}}) &= \frac{d}{d\tilde{s}_{\mathbf{q}}(\hat{\mathbf{x}})}\left[1 - \chi^2_{2n}(-2\log(\tilde{s}_{\mathbf{q}}(\hat{\mathbf{x}})))\right] \cdot \frac{\partial}{\partial q_k}\tilde{s}_{\mathbf{q}}(\hat{\mathbf{x}}) \\ \frac{d}{d\tilde{s}_{\mathbf{q}}(\hat{\mathbf{x}})}\left[1 - \chi^2_{2n}(-2\log(\tilde{s}_{\mathbf{q}}(\hat{\mathbf{x}})))\right] &= \frac{1}{(n-1)!}(-\log(\tilde{s}_{\mathbf{q}}(\hat{\mathbf{x}})))^{n-1} \quad .\end{aligned}$$

The second derivative is non-negative since $0 \leq \tilde{s}_{\mathbf{q}}(\hat{\mathbf{x}}) \leq 1$. Further, the partial derivative of $\tilde{s}_{\mathbf{q}}(\hat{\mathbf{x}})$ with respect to q_k is simply $\frac{\partial}{\partial q_k}\tilde{s}_{\mathbf{q}}(\hat{\mathbf{x}}) = \prod_{i \neq k : \hat{x}_i = 1} q_i \geq 0$. Thus, for all k,

$$\frac{\partial}{\partial q_k}\tilde{S}(\hat{\mathbf{x}}) \geq 0 \quad .$$

By an analogous derivation, replacing q_i by $1 - q_i$,

$$\frac{\partial}{\partial q_k}\tilde{H}(\hat{\mathbf{x}}) \leq 0 \quad .$$

The final result is then give by

$$\frac{\partial}{\partial q_k}\tilde{I}(\hat{\mathbf{x}}) = \frac{1}{2}\frac{\partial}{\partial q_k}\tilde{S}(\hat{\mathbf{x}}) - \frac{1}{2}\frac{\partial}{\partial q_k}\tilde{H}(\hat{\mathbf{x}}) \geq 0 \quad .$$

□

However, unlike the simple products, the statistics $s_{\mathbf{q}}(\cdot)$ and $h_{\mathbf{q}}(\cdot)$ have unusual behavior because the function $\boldsymbol{\delta}(\cdot)$ sanitizes the token scores. Namely, $\boldsymbol{\delta}(\cdot)$ is the indicator function of the set $\mathbb{T}_{\hat{\mathbf{x}}}$. Membership in this set is determined by absolute distance of a token's score from the agnostic score of $\frac{1}{2}$; *i.e.*, by the value $g_i \triangleq \left|q_i - \frac{1}{2}\right|$. The i^{th} token belongs to $\mathbb{T}_{\hat{\mathbf{x}}}$ if *i)* $\hat{x}_i = 1$ *ii)* $g_i \geq Q$ (by default $Q = 0.1$ so all tokens in $(0.4, 0.6)$ are excluded) and *iii)* of the remaining tokens, the token has among the largest T values of g_i (by default $T = 150$).

For my purposes, for every message $\hat{\mathbf{x}}$, there is some value $\tau_{\hat{\mathbf{x}}} < \frac{1}{2}$ that defines an interval $\left(\frac{1}{2} - \tau_{\hat{\mathbf{x}}}, \frac{1}{2} + \tau_{\hat{\mathbf{x}}}\right)$ to exclude tokens. That is

$$\delta(\hat{\mathbf{x}})_i = \hat{x}_i \cdot \begin{cases} 0 & \text{if } q_i \in \left(\frac{1}{2} - \tau_{\hat{\mathbf{x}}}, \frac{1}{2} + \tau_{\hat{\mathbf{x}}}\right) \\ 1 & \text{otherwise} \end{cases} \quad .$$

Clearly, for tokens in $\hat{\mathbf{x}}$, $\delta(\hat{\mathbf{x}})_i$ steps from 1 to 0 and back to 1 as q_i increases. This causes $s_{\mathbf{q}}(\hat{\mathbf{x}})$ to have two discontinuities with respect to q_i: it increases linearly on the intervals $\left[0, \frac{1}{2} - \tau_{\hat{\mathbf{x}}}\right]$ and $\left[\frac{1}{2} + \tau_{\hat{\mathbf{x}}}, 1\right]$, but on the middle interval $\left(\frac{1}{2} - \tau_{\hat{\mathbf{x}}}, \frac{1}{2} + \tau_{\hat{\mathbf{x}}}\right)$ it jumps discontinuously to its maximum value. This behavior of is depicted in Figure B.1. Similarly, $h_{\mathbf{q}}(\hat{\mathbf{x}})$ decreases linearly except on the middle interval $\left(\frac{1}{2} - \tau_{\hat{\mathbf{x}}}, \frac{1}{2} + \tau_{\hat{\mathbf{x}}}\right)$ where it also jumps to its maximum value. Thus, neither $s_{\mathbf{q}}(\hat{\mathbf{x}})$ or $h_{\mathbf{q}}(\hat{\mathbf{x}})$ have monotonic behavior on the interval $[0, 1]$.

To better understand how $I(\hat{\mathbf{x}})$ behaves when q_i increases given that neither $s_{\mathbf{q}}(\hat{\mathbf{x}})$ or $h_{\mathbf{q}}(\hat{\mathbf{x}})$ are monotonic, I analyze its behavior on a case by case basis. For this purpose, I refer to the three intervals $\left[0, \frac{1}{2} - \tau_{\hat{\mathbf{x}}}\right]$, $\left(\frac{1}{2} - \tau_{\hat{\mathbf{x}}}, \frac{1}{2} + \tau_{\hat{\mathbf{x}}}\right)$, and $\left[\frac{1}{2} + \tau_{\hat{\mathbf{x}}}, 1\right]$ as $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$,

respectively. Clearly, if q_i increases but stays within the same interval, $I(\hat{\mathbf{x}})$ also increases. This follows from Lemma B.1 and the fact that $I(\hat{\mathbf{x}})$ will not change if q_i remains within interval $\mathbb{B}$. Similarly, $I(\hat{\mathbf{x}})$ also increases if q_i increases from interval $\mathbb{A}$ to interval $\mathbb{C}$; this too follows from Lemma B.1. The only cases when $I(\hat{\mathbf{x}})$ may decrease when q_i increases occur when either q_i transitions from interval $\mathbb{A}$ to $\mathbb{B}$ or q_i transitions from interval $\mathbb{B}$ to $\mathbb{C}$, but in these cases, the behavior of $I(\hat{\mathbf{x}})$ depends heavily on the scores for the other tokens in $\hat{\mathbf{x}}$ and the value of q_i before it increases as depicted by Figure B.2. It is also worth noting that $I(\hat{\mathbf{x}})$ in fact will *never* decrease if $\hat{\mathbf{x}}$ has more than 150 tokens outside the interval $(0.4, 0.6)$, since in this case increasing q_i either into or out of $\mathbb{B}$ also corresponds to either adding or removing a second token score q_j. The effect in this case is that $I(\hat{\mathbf{x}})$ always increases.

The attacks against SpamBayes that I introduce in Chapter 4.3 ignore the fact that $I(\hat{\mathbf{x}})$ may decrease when increasing some token scores. In this sense, these attacks are not truly optimal. However, determining which set of tokens will optimally increase the overall $I(\cdot)$ of a set of future messages $\{\hat{\mathbf{x}}\}$ is a combinatorial problem that seems infeasible for a real-world adversary. Instead, I consider attacks that are optimal for the relaxed version of the problem that incorporates *all* tokens from $\hat{\mathbf{x}}$ in computing $I(\hat{\mathbf{x}})$. Further, in Chapter 4.5, I show that these approximate techniques are extraordinarily effective against SpamBayes in spite of the fact some non-optimal tokens are included in the attack messages.

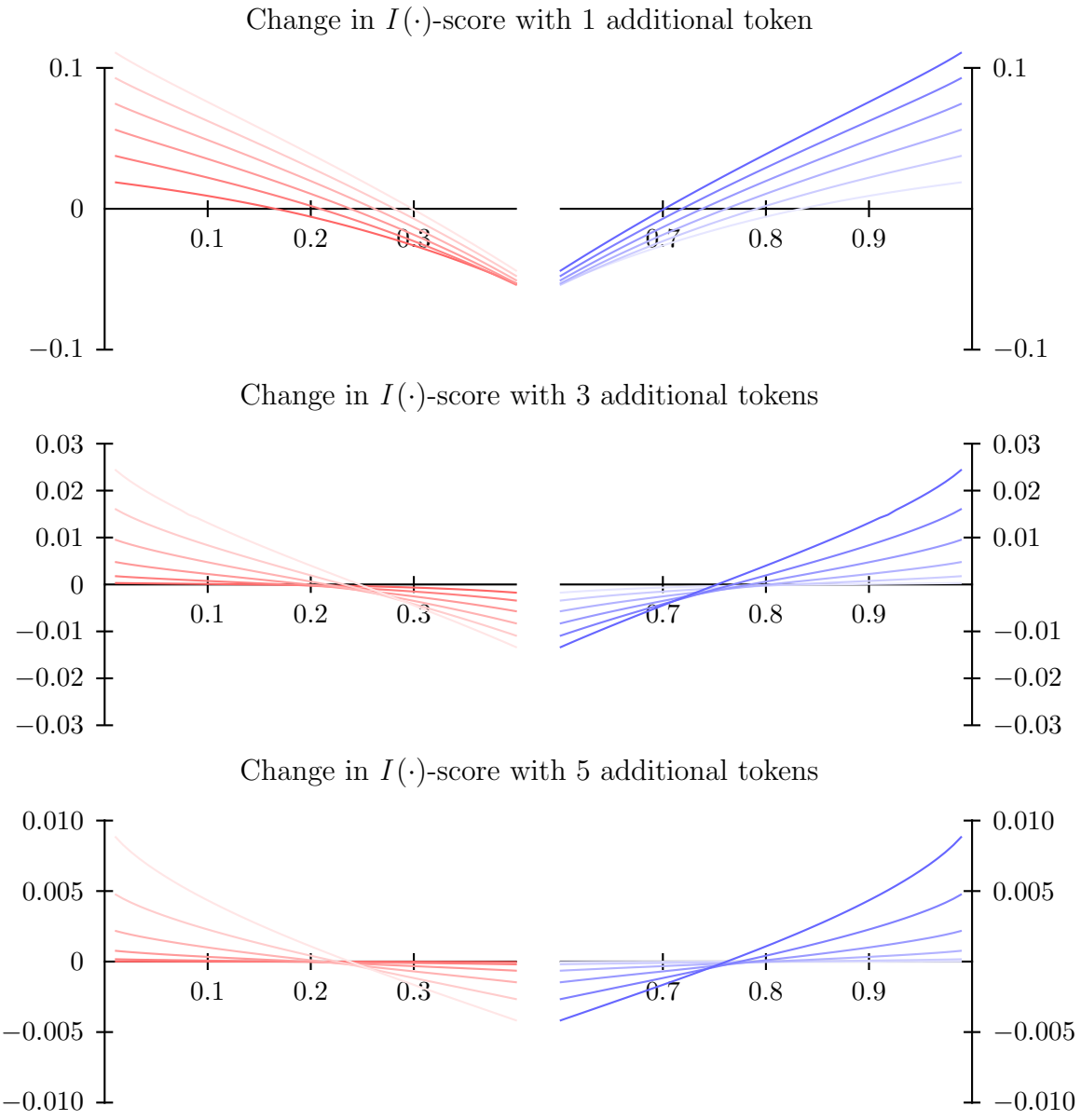


Figure B.2: The effect of the $\boldsymbol{\delta}(\cdot)$ function on $I(\cdot)$ as the score of the i^{th} token, q_i, increases causing q_i to move into or out of the region $(0.4, 0.6)$ where all tokens are ignored. In each plot, the x-axis is the value of q_i before it's removal and the y-axis is the change in $I(\cdot)$ due to the removal; note that the scale on the y-axis decreases from top to bottom. For the top-most row of plots there is 1 unchanged token scores in addition to the changing one, for the middle row there are 3 additional unchanged token scores, and for the bottom row there are 5 additional unchanged token scores. The plots in the left-most column demonstrate the effect of removing the i^{th} token when initially $q_i \in (0, 0.4)$; the scores of the additional unchanging tokens are all fixed to the same value of 0.02 (dark red), 0.04, 0.06, 0.08, 0.10, or 0.12 (light red). The plots in the right-most column demonstrate the effect of adding the i^{th} token when initially $q_i \in (0.4, 0.6)$; the scores of the additional unchanging tokens are all fixed to the same value of 0.88 (dark blue), 0.90, 0.92, 0.94, 0.96, or 0.98 (light blue).

Appendix C

Proofs for Near-Optimal Evasion

In this appendix, I give proofs for the theorems from Chapter 6. First, I show that the query complexity of K-step MultiLineSearch is $\mathcal{O}\left(L_\epsilon + \sqrt{L_\epsilon}|\mathbb{W}|\right)$ when $K = \lceil\sqrt{L_\epsilon}\rceil$. Second, I show three lower bounds for different cost functions. Each of the lower bound proofs follow a similar argument: I use classifiers based on the cost-ball and classifiers based on the convex hull of the queries to construct two alternative classifiers with different ϵ-*IMAC*s. This allows me to show results on the minimal number of queries required.

C.1 Proof of K-step MultiLineSearch Theorem

To analyze the worst case of K-step MultiLineSearch (Algorithm 6.4), I analyze the malicious classifier that maximizes the number of queries. I refer to the querier as the *adversary*.

Proof of Theorem 6.3. At each each iteration of Algorithm 6.4, the adversary chooses some direction, $\mathbf{e}$ not yet eliminated from $\mathbb{W}$. Every direction in $\mathbb{W}$ is feasible (*i.e.*, could yield an ϵ-*IMAC*) and the malicious classifier, by definition, will make this choice as costly as possible. During the K steps of binary search along this direction, regardless of which direction $\mathbf{e}$ is selected or how the malicious classifier responds, the candidate multiplicative gap (see Section 6.1.3) along $\mathbf{e}$ will shrink by an exponent of 2^{-K}; *i.e.*,

$$\frac{B^-}{B^+} = \left(\frac{C^-}{C^+}\right)^{2^{-K}} \tag{C.1}$$

$$\log\left(G'_{t+1}\right) = \log\left(G_t\right) \cdot 2^{-K} \tag{C.2}$$

The primary decision for the malicious classifier occurs when the adversary begins querying other directions beside $\mathbf{e}$. At iteration t, the malicious classifier has 2 options:

Case 1 ($t \in \mathbb{C}_1$): Respond with '+' for all remaining directions. Here the bound candidates B^+ and B^- are verified and thus the new gap is reduced by an exponent of 2^{-K}; however, no directions are eliminated from the search.

Case 2 ($t \in \mathbb{C}_2$): Choose at least 1 direction to respond with '$-$'. Here since only the value of C^- changes, the malicious classifier can choose to respond to the first K queries so that the gap decreases by a negligible amount (by always responding with '+' during the first K queries along $\mathbf{e}$, the gap only decreases by an exponent of $(1 - 2^{-K})$). However, the malicious classifier must chose some number $E_t \geq 1$ of directions that will be eliminated.

By conservatively assuming the gap only decreases in case 1, the total number of queries is bounded for both cases independent of the order in which the malicious classifier applies them.

At the t^{th} iteration, the malicious classifier can either decide to be in case 1 ($t \in \mathbb{C}_1$) or case 2 ($t \in \mathbb{C}_2$). I assume that the gap only decreases in the case 1. That is, I define $G_0 = C_0^-/C_0^+$ so that if $t \in \mathbb{C}_1$, then $G_t = G_{t-1}^{2^{-K}}$ whereas if $t \in \mathbb{C}_2$, then $G_t = G_{t-1}$. This assumption yields an upper bound on the algorithm's performance and decouples the analysis of the queries for $\mathbb{C}_1$ and $\mathbb{C}_2$. From it, I derive the following upper bound on the number of case 1 iterations that must occur before our algorithm terminates; simply stated, there must be a total of at least L_ϵ binary search steps made during the case 1 iterations and every case 1 iteration makes exactly K steps. More formally, each case 1 iteration reduces the gap by an exponent of 2^{-K} and our termination condition is $G_T \leq 1 + \epsilon$. Since our algorithm will terminate as soon as the gap $G_T \leq 1 + \epsilon$, iteration T must be a case 1 iteration and $G_{T-1} > 1 + \epsilon$ (otherwise the algorithm would have terminated earlier). From this the total number of iterations must satisfy

$$\begin{aligned}
\log_2(G_{T-1}) &> \log_2(1+\epsilon) \\
\underbrace{\log_2(G_0) \prod_{i \in C_1 \wedge i < T} 2^{-K}}_{\text{by Equation (C.2)}} &> \log_2(1+\epsilon) \\
2^{-\sum_{i \in C_1 \wedge i < T} K} &> \frac{\log_2(1+\epsilon)}{\log_2(G_0)} \\
\sum_{i \in C_1 \wedge i < T} K &> \underbrace{\log_2 \frac{\log_2(G_0)}{\log_2(1+\epsilon)}}_{=L_\epsilon \text{ by Equation (6.6)}} \\
(|C_1| - 1)K &< L_\epsilon
\end{aligned}$$

where the factor $(|\mathbb{C}_1| - 1)$ comes as a result of excluding the last case 1 iteration, T. A similar derivation for $G_T \leq 1 + \epsilon$ yields $|\mathbb{C}_1| \cdot K \geq L_\epsilon$ and the only integer that satisfies both these conditions is:

$$|\mathbb{C}_1| = \left\lceil \frac{L_\epsilon}{K} \right\rceil \quad . \tag{C.3}$$

Now, at every case 1 iteration, the adversary make exactly $K + |\mathbb{W}_t| - 1$ queries where $\mathbb{W}_t$ is the set of feasible directions remaining at the t^{th} iteration. While $\mathbb{W}_t$ is controlled by the malicious classifier, it is bounded by $|\mathbb{W}_t| \leq |\mathbb{W}|$. Using this and the relation from

Equation (C.3), I bound the number of queries Q_1 used in case 1 by

$$\begin{aligned} Q_1 &= \sum_{t \in C_1} (K + |\mathbb{W}_t| - 1) \\ &\leq \sum_{t \in C_1} (K + |\mathbb{W}| - 1) \\ &= \left\lceil \frac{L_\epsilon}{K} \right\rceil \cdot (K + |\mathbb{W}| - 1) \\ &\leq \left(\frac{L_\epsilon}{K} + 1 \right) \cdot K + \left\lceil \frac{L_\epsilon}{K} \right\rceil \cdot (|\mathbb{W}| - 1) \\ &= L_\epsilon + K + \left\lceil \frac{L_\epsilon}{K} \right\rceil \cdot (|\mathbb{W}| - 1) \quad . \end{aligned}$$

For each case 2 iteration, the adversary makes exactly $K + E_t$ queries and this causes the elimination of $E_t \geq 1$ directions; hence, $|\mathbb{W}_{t+1}| = |\mathbb{W}_t| - E_t$. The malicious classifier will always make $E_t = 1$ in every case 2 instance since that maximally limits how much the adversary gains. Nevertheless, since case 2 requires the elimination of at least 1 direction, the following bound applies: $|\mathbb{C}_2| \leq |\mathbb{W}| - 1$. Moreover, regardless of the choice of E_t, $\sum_{t \in \mathbb{C}_2} E_t \leq |\mathbb{W}| - 1$ since each direction can be eliminated no more than once. Thus,

$$\begin{aligned} Q_2 &= \sum_{i \in \mathbb{C}_2} (K + E_t) \\ &\leq |\mathbb{C}_2| \cdot K + |\mathbb{W}| - 1 \\ &\leq (|\mathbb{W}| - 1)(K + 1) \quad . \end{aligned}$$

The total number of queries used by Algorithm 6.4

$$\begin{aligned} Q = Q_1 + Q_2 &\leq L_\epsilon + K + \left\lceil \frac{L_\epsilon}{K} \right\rceil \cdot (|\mathbb{W}| - 1) + (|\mathbb{W}| - 1)(K + 1) \\ &= L_\epsilon + \left\lceil \frac{L_\epsilon}{K} \right\rceil \cdot |\mathbb{W}| + K \cdot |\mathbb{W}| + |\mathbb{W}| - \left\lceil \frac{L_\epsilon}{K} \right\rceil - 1 \\ &= L_\epsilon + \left(\left\lceil \frac{L_\epsilon}{K} \right\rceil + K + 1 \right) |\mathbb{W}| \end{aligned}$$

Finally, choosing $K = \lceil \sqrt{L_\epsilon} \rceil$ minimizes this expression. By substituting this K into Q's bound and using the bound $L_\epsilon / \lceil \sqrt{L_\epsilon} \rceil \leq \sqrt{L_\epsilon}$, yields

$$Q \leq L_\epsilon + \left(2 \lceil \sqrt{L_\epsilon} \rceil + 1 \right) |\mathbb{W}|$$

so $Q = \mathcal{O}\left(L_\epsilon + \sqrt{L_\epsilon} |\mathbb{W}| \right)$. □

C.2 Proof of Lower Bounds

Here I give proofs for the lower bound theorems in Section 6.2.1.2 first giving the proof for the more complicated multiplicative case followed by a similar proof sketch for the additive

case. For these lower bounds, D is the dimension of the space, $A : \Re^D \mapsto \Re^+$ is any positive convex function, $0 < C_0^+ < C_0^-$ are initial upper and lower bounds on the MAC, and $\hat{\mathcal{F}}^{\text{convex,'+'}} \subset \mathcal{F}^{\text{convex,'+'}}$ is the set of classifiers consistent with the constraints on the MAC; *i.e.*, for $f \in \hat{\mathcal{F}}^{\text{convex,'+'}}$ the set $\mathcal{X}_f^+$is convex, $\mathbb{B}^{C_0^+}(A) \subset \mathcal{X}_f^+$ and $\mathbb{B}^{C_0^-}(A) \not\subset \mathcal{X}_f^+$.

Proof of Theorem 6.5 and 6.4. Suppose a query-based algorithm submits $N < D+1$ membership queries $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)} \in \Re^D$ to the classifier. For the algorithm to be ϵ-optimal, these queries must constrain all consistent classifiers is the family $\hat{\mathcal{F}}^{\text{convex,'+'}}$ to have a common point among their ϵ-$IMAC$ sets. Suppose that the responses to the queries are consistent with the classifier f defined as:

$$f(\mathbf{x}) = \begin{cases} +1\,, & \text{if } A(\mathbf{x}) < C_0^- \\ -1\,, & \text{otherwise} \end{cases} . \tag{C.4}$$

For this classifier, $\mathcal{X}_f^+$ is convex since A is a convex function, $\mathbb{B}^{C_0^+}(A) \subset \mathcal{X}_f^+$ since $C_0^+ < C_0^-$, and $\mathbb{B}^{C_0^-}(A) \not\subset \mathcal{X}_f^+$ since $\mathcal{X}_f^+$ is the open C_0^--ball whereas $\mathbb{B}^{C_0^-}(A)$ is the closed C_0^--ball. Moreover, since $\mathcal{X}_f^+$ is the open C_0^--ball, $\nexists\, \mathbf{x} \in \mathcal{X}_f^-$ suchthat $A(\mathbf{x}) < C_0^-$ therefore $MAC(f, A) = C_0^-$, and any ϵ-optimal points $\mathbf{x}' \in \epsilon\text{-}IMAC^{(*)}(f, A)$ must satisfy $C_0^- \leq A(\mathbf{x}') \leq (1+\epsilon)C_0^-$. Similarly, any η-optimal points $\mathbf{x}' \in \eta\text{-}IMAC^{(+)}(f, A)$ must satisfy $C_0^- \leq A(\mathbf{x}') \leq C_0^- + \eta$.

Consider an alternative classifier g that responds identically to f for $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$ but has a different convex positive set $\mathcal{X}_g^+$. Without loss of generality, suppose the first $M \leq N$ queries are positive and the remaining are negative. Let $\mathbb{G} = conv\left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)}\right)$; that is, the convex hull of the M positive queries. Now let $\mathcal{X}_g^+$ be the convex hull of $\mathbb{G}$ and the C_0^+-ball of A: $\mathcal{X}_g^+ = conv\left(\mathbb{G} \cup \mathbb{B}^{C_0^+}(A)\right)$. Since $\mathbb{G}$ contains all positive queries and $C_0^+ < C_0^-$, the convex set $\mathcal{X}_g^+$ is consistent with the observed responses, $\mathbb{B}^{C_0^+}(A) \subset \mathcal{X}_g^+$ by definition, and $\mathbb{B}^{C_0^-}(A) \not\subset \mathcal{X}_g^+$ since the positive queries are all inside the open C_0^--sublevel set. Further, since $M \leq N < D+1$, $\mathbb{G}$ is contained in a proper linear subspace of $\Re^D$ and hence $int(\mathbb{G}) = \emptyset$. Hence, there is always some point from $\mathbb{B}^{C_0^+}(A)$ that is on the boundary of $\mathcal{X}_g^+$; *i.e.*, $\mathbb{B}^{C_0^+}(A) \not\subset int(\mathbb{G})$ because $int(\mathbb{G}) = \emptyset$, hence, there must be at least one point from $\mathbb{B}^{C_0^+}(A)$ on the boundary of the convex hull of $\mathbb{B}^{C_0^+}(A)$ and $\mathbb{G}$. Hence, $MAC(g, A) = \inf_{\mathbf{x} \in \mathcal{X}_g^-}[A(\mathbf{x})] = C_0^+$. Since the accuracy $\epsilon < \frac{C_0^-}{C_0^+} - 1$, any $\mathbf{x} \in \epsilon\text{-}IMAC^{(*)}(g, A)$ must have

$$A(\mathbf{x}) \leq (1+\epsilon)C_0^+ < \frac{C_0^-}{C_0^+}C_0^+ = C_0^-$$

whereas any $\mathbf{y} \in \epsilon\text{-}IMAC^{(*)}(f, A)$ must have $A(\mathbf{y}) \geq C_0^-$. Thus, $\epsilon\text{-}IMAC^{(*)}(f, A) \cap \epsilon\text{-}IMAC^{(*)}(g, A) = \emptyset$ and we have constructed two convex-inducing classifiers f and g both consistent with the query responses with no common ϵ-$IMAC^{(*)}$. Similarly, since $\eta < C_0^- - C_0^+$, any $\mathbf{x} \in \eta\text{-}IMAC^{(+)}(g, A)$ must have

$$A(\mathbf{x}) \leq \eta + C_0^+ < C_0^- - C_0^+ + C_0^+ = C_0^-$$

whereas any $\mathbf{y} \in \eta\text{-}IMAC^{(+)}(f, A)$ must have $A(\mathbf{y}) \geq C_0^-$. Thus, $\eta\text{-}IMAC^{(+)}(f, A) \cap \eta\text{-}IMAC^{(+)}(g, A) = \emptyset$ and so the two convex-inducing classifiers f and g also have no common η-$IMAC^{(+)}$.

Suppose instead that a query-based algorithm submits $N < L_\epsilon^{(*)}$ membership queries (or $N < L_\eta^{(+)}$ for the additive case). Recall our definitions: C_0^- is the initial upper bound on the MAC, C_0^+ is the initial lower bound on the MAC, and $G_t^{(*)} = C_t^-/C_t^+$ is the gap between the upper bound and lower bound at iteration t ($G_t^{(+)} = C_t^- - C_t^+$ for the additive case). Here, the malicious classifier f responds with

$$f\left(\mathbf{x}^{(t)}\right) = \begin{cases} +1\,, & \text{if } A\left(\mathbf{x}^{(t)}\right) \leq \sqrt{C_{t-1}^- \cdot C_{t-1}^+} \\ -1\,, & \text{otherwise} \end{cases}. \tag{C.5}$$

When the classifier responds with '+', C_t^+ increases to no more than $\sqrt{C_{t-1}^- \cdot C_{t-1}^+}$ and so $G_t \geq \sqrt{G_{t-1}}$. Similarly when this classifier responds with '−', C_t^- decreases to no less than $\sqrt{C_{t-1}^- \cdot C_{t-1}^+}$ and so again $G_t \geq \sqrt{G_{t-1}}$. Thus, these responses ensure that at each iteration $G_t \geq \sqrt{G_{t-1}}$ and since the algorithm can not terminate until $G_N \leq 1+\epsilon$, which yields $N \geq L_\epsilon^{(*)}$ from Equation (6.6) (or in the additive case $N \geq L_\eta^{(+)}$ from Equation (6.5)). Again, I have constructed two convex-inducing classifiers with consistent query responses but with no common ϵ-$IMAC$. The first classifier's positive set is the smallest cost-ball enclosing all positive queries, while the second classifier's positive set is the largest cost-ball enclosing all positive queries but no negatives. The MAC values of these sets differ by more than a factor of $(1+\epsilon)$ if $N < L_\epsilon^{(*)}$ (or, for the additive case, by a difference of more than η if $N < L_\eta^{(+)}$), so they have no common ϵ-$IMAC$. . □

C.3 Proof of Theorem 6.9

For the proof of Theorem 6.9, I use the orthants (centered at $\mathbf{x}^A$)—*i.e.*, an *orthant* is the D-dimensional generalization of a quadrant in 2-dimensions. There are 2^D orthants in a D-dimensional space. I represent each orthant by it's *canonical representation* which is a vector of D positive or negative ones; *i.e.* the orthant represented by $\mathbf{a} = \langle \pm 1, \pm 1, \ldots, \pm 1 \rangle$ contains the point $\mathbf{x}^A + \mathbf{a}$ and is the set of all points $\mathbf{x}$ satisfying:

$$x_i \in \begin{cases} [0, +\infty]\,, & \text{if } a_i = +1 \\ [-\infty, 0]\,, & \text{if } a_i = -1 \end{cases}.$$

Now based on Lemma A.2 from Appendix A.2, I give the required proof of Theorem 6.9:

Proof of Theorem 6.9. Suppose a query-based algorithm submits N membership queries $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)} \in \Re^D$ to the classifier. Again, for the algorithm to be ϵ-optimal, these queries must constrain all consistent classifiers in the family $\hat{\mathcal{F}}^{\text{convex,'+'}}$ to have a common point among their ϵ-$IMAC$ sets. The responses described above are consistent with the classifier f defined as

$$f(\mathbf{x}) = \begin{cases} +1\,, & \text{if } A_p(\mathbf{x}) < C_0^- \\ -1\,, & \text{otherwise} \end{cases}.$$

For this classifier, $\mathcal{X}_f^+$ is convex since A_p is a convex function for $p \geq 1$, $\mathbb{B}^{C_0^+}(A_p) \subset \mathcal{X}_f^+$ since $C_0^+ < C_0^-$, and $\mathbb{B}^{C_0^-}(A_p) \not\subset \mathcal{X}_f^+$ since $\mathcal{X}_f^+$ is the open C_0^--ball whereas $\mathbb{B}^{C_0^-}(A_p)$ is the

closed C_0^--ball. Moreover, since $\mathcal{X}_f^+$ is the open C_0^--ball, $\nexists\, \mathbf{x} \in \mathcal{X}_f^-$ suchthat $A_p(\mathbf{x}) < C_0^-$ therefore $MAC(f, A_p) = C_0^-$, and any ϵ-optimal points $\mathbf{x}' \in \epsilon\text{-}IMAC^{(*)}(f, A_p)$ must satisfy $C_0^- \leq A_p(\mathbf{x}') \leq (1+\epsilon)C_0^-$.

Now consider an alternative classifier g that responds identically to f for $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$ but has a different convex positive set $\mathcal{X}_g^+$. Without loss of generality suppose the first $M \leq N$ queries are positive and the remaining are negative. Here, consider a set which is a convex hull of the orthants of all M positive queries; that is,

$$\mathbb{G} = conv\left(orth\left(\mathbf{x}^{(1)}\right) \cap \mathcal{X}_f^+, orth\left(\mathbf{x}^{(2)}\right) \cap \mathcal{X}_f^+, \ldots, orth\left(\mathbf{x}^{(M)}\right) \cap \mathcal{X}_f^+\right) \tag{C.6}$$

where $orth(\mathbf{x})$ is some orthant that $\mathbf{x}$ lies with in relative to $\mathbf{x}^A$ (a data point may lie within more than one orthant but it is only necessary to select one of the orthants that contains it to cover it). By intersecting each data point's orthant with the set $\mathcal{X}_f^+$ and taking the convex hull of these regions, $\mathbb{G}$ is convex , contains $\mathbf{x}^A$ and is a subset of $\mathcal{X}_f^+$ that is also consistent with all the query responses of f; *i.e.*, each of the M positive queries are in $\mathcal{X}_g^+$ and all the negative queries are in $\mathcal{X}_g^-$. Moreover, $\mathbb{G}$ is a superset of the convex hull of the M positive queries. Thus, the largest enclosed ℓ_p ball within the $\mathbb{G}$ is an upper bound on $MAC(g, A_p)$, so I bound the size of this ℓ_p ball instead.

I now represent each orthant as a vertex in a D-dimensional hypercube graph—the Hamming distance between any pair of orthants is the number of different coordinates in their canonical representations and two orthants are adjacent in the graph if and only if they have Hamming distance of one. Using this notion of Hamming distance, I find a K-covering of the hypercube. I refer to the orthants used to construct $\mathbb{G}$ in Equation C.6 as *covering orthants* because they cover the M positive queries. The vertices corresponding to these covering orthants form a covering of the hypercube. Suppose the M covering orthants are sufficient for a K covering but not $K-1$ covering; then there must be at least one vertex not in the covering that has at least a K Hamming distance to every vertex in the covering. This vertex corresponds to an empty orthant that differs from all covered orthants in at least K coordinates of their canonical vertices. Without loss of generality, suppose this uncovered orthant has the canonical vertex of all positive ones which is scaled to $C_0^-\mathbf{1}$. Now, consider the hyperplane with normal vector $\mathbf{w} = \mathbf{1}$ and displacement

$$d = \begin{cases} C_0^-(D-K)^{\frac{p-1}{p}} & \text{if } 1 < p < \infty \\ C_0^-(D-K) & \text{if } p = \infty \end{cases}$$

that specifies the discriminant function $s(\mathbf{x}) = \mathbf{x}^\top\mathbf{w} - d = \sum_{i=1}^D x_i - d$. For this hyperplane, the vertex $C_0^-\mathbf{1}$ yields

$$\begin{aligned} s\left(C_0^-\mathbf{1}\right) &= C_0^- D - d \\ &= C_0^- D - \left(C_0^- D - K\right)^{\frac{p-1}{p}} \\ &> C_0^- D - \left(C_0^- D - K\right) \\ &> 0\ . \end{aligned}$$

Also for any orthant $\mathbf{a}$ with Hamming distance at least K from this uncovered orthant, all points $\mathbf{x} \in orth(\mathbf{a}) \cap \mathcal{X}_f^+$ yield the following valuation of the function s, by definition of the

orthant and $\mathcal{X}_f^+$:

$$\begin{aligned} s(\mathbf{x}) &= \sum_{i=1}^{D} x_i - d \\ &= \sum_{\{i \mid a_i=+1\}} \underbrace{x_i}_{\geq 0} + \sum_{\{i \mid a_i=-1\}} \underbrace{x_i}_{\leq 0} - d \ . \end{aligned}$$

Since all the terms in the second summation are non-positive, the second sum is at most 0. Thus, maximizing the first summation upper bounds $s(\mathbf{x})$. The summation $\sum_{\{i \mid a_i=+1\}} x_i$ (with the constraint that $\|\mathbf{x}\|_p < C_0^-$ which is necessary for $\mathbf{x}$ to be in $\mathcal{X}_f^+$) has at most $D-K$ terms and is maximized by $x_i = C_0^-(D-K)^{-1/p}$ (or $x_i = C_0^-$ for $p=\infty$) for which the first summation is upper bounded by $C_0^-(D-K)^{\frac{p-1}{p}}$ or $C_0^-(D-K)$ for $p=\infty$; *i.e.* it is upper bounded by d and so $s(\mathbf{x}) \leq 0$. Thus, this hyperplane separates the scaled vertex $C_0^-\mathbf{1}$ from each set $orth(\mathbf{a}) \cap \mathcal{X}_f^+$ where $\mathbf{a}$ is the canonical representation of any orthant with a Hamming distance of at least K from the positive orthant represented by $\mathbf{1}$. This hyperplane also separates the scaled vertex from $\mathbb{G}$ by the properties of the convex hull. Since the displacement d defined above is greater than 0, by applying Lemma A.3, this separating hyperplane upper bounds the cost of the largest ℓ_p ball enclosed in $\mathbb{G}$ as

$$MAC(g, A_p) \leq C_0^-(D-K)^{\frac{p-1}{p}} \cdot \|\mathbf{1}\|_{\frac{p}{p-1}}^{-1} = C_0^- \left(\frac{D-K}{D}\right)^{\frac{p-1}{p}}$$

for $1 < p < \infty$ and

$$MAC(g, A_p) \leq C_0^-(D-K) \cdot \|\mathbf{1}\|_1^{-1} = C_0^- \frac{D-K}{D}$$

for $p=\infty$. Based on this upper bound on the MAC of g and the MAC of f (*i.e.*, C_0^-), if there is a common ϵ-$IMAC$ between these classifiers, it must satisfy

$$(1+\epsilon) \geq \begin{cases} \left(\frac{D}{D-K}\right)^{\frac{p-1}{p}}, & \text{if } 1 < p < \infty \\ \frac{D}{D-K}, & \text{if } p = \infty \end{cases} .$$

Solving for the value of K required to achieve a desired accuracy of $1+\epsilon$ yields

$$K \leq \begin{cases} \frac{(1+\epsilon)^{\frac{p}{p-1}}-1}{(1+\epsilon)^{\frac{p}{p-1}}} D, & \text{if } 1 < p < \infty \\ \frac{\epsilon}{1+\epsilon} D, & \text{if } p = \infty \end{cases} ,$$

which bounds the size of the covering required to achieve the desired multiplicative accuracy ϵ.

For the case $1 < p < \infty$, Lemma A.2 shows there must be

$$M \geq \exp\left\{\ln(2) \cdot D\left(1 - H\left(1 - (1+\epsilon)^{\frac{p}{1-p}}\right)\right)\right\}$$

vertices of the hypercube in the covering to achieve any desired accuracy $0 < \epsilon < 2^{\frac{p-1}{p}} - 1$, for which

$$\delta = \frac{(1+\epsilon)^{\frac{p}{p-1}} - 1}{(1+\epsilon)^{\frac{p}{p-1}}} < \frac{1}{2}$$

to satisfy the condition required by the lemma. Thus, this theorem is applicable for any ϵ that satisfies,

$$\epsilon < 2^{\frac{p-1}{p}} - 1 \ .$$

For example, for $p = 2$, the theorem is applicable for any $\epsilon < \sqrt{2} - 1$. Moreover, since $0 < H(\delta) < 1$ for any $0 < \delta < 1$,

$$\alpha_{p,\epsilon} \quad = \quad \exp\left\{\ln(2)\left(1 - H\left(\frac{(1+\epsilon)^{\frac{p}{p-1}} - 1}{(1+\epsilon)^{\frac{p}{p-1}}}\right)\right)\right\} > 1$$

and

$$M > \alpha_{p,\epsilon}^{D} \ .$$

Similarly for $p = \infty$, applying Lemma A.2 requires

$$M \geq 2^{D\left(1 - H\left(\frac{\epsilon}{1+\epsilon}\right)\right)}$$

to achieve any desired accuracy $0 < \epsilon < 1$ (for which $\epsilon/(1+\epsilon) < 1/2$ as required by the lemma). Again, by the properties of entropy the constant $\alpha_{\infty,\epsilon} = 2^{\left(1 - H\left(\frac{\epsilon}{1+\epsilon}\right)\right)} > 1$ for any $0 < \epsilon < 1$ and $M > \alpha_{\infty,\epsilon}^{D}$. □

It is worth noting that the constants $\alpha_{p,\epsilon}$ and $\alpha_{\infty,\epsilon}$ required by Theorem 6.9 can be expressed in a more concise form by expanding the entropy function ($H(\delta) = -\delta \log_2(\delta) - (1-\delta)\log_2(1-\delta)$). For $1 < p < \infty$ the constant is given by

$$\alpha_{p,\epsilon} = 2 \cdot \left(1 - (1+\epsilon)^{\frac{p}{1-p}}\right) \cdot \exp\left(\ln\left(\frac{-1}{1 - (1+\epsilon)^{\frac{p}{p-1}}}\right) \cdot (1+\epsilon)^{\frac{p}{1-p}}\right) \ . \tag{C.7}$$

In this form, it is difficult to directly see that $\alpha_{p,\epsilon} > 1$ for $\epsilon < 2^{\frac{p-1}{p}} - 1$, but using the entropy form in the proof above shows that this is indeed the case. Similarly, for $p = \infty$ the more concise form of the constant is given by

$$\alpha_{\infty,\epsilon} = \frac{2}{1+\epsilon} \exp\left(\ln(\epsilon) \cdot \left(\frac{\epsilon}{1+\epsilon}\right)\right) \ . \tag{C.8}$$

Again, as shown in the proof above, $\alpha_{\infty,\epsilon} > 1$ for $\epsilon < 1$.

C.4 Proof of Theorem 6.10

For this proof, I build on previous results for covering hyperspheres. The proof is based on a covering number result from Wyner [1965] that first appeared in Shannon [1959]. This result bounds the minimum number of spherical caps required to cover the surface of a hypersphere and is summarized in Appendix A.1.

Proof of Theorem 6.10. Suppose a query-based algorithm submits $N < D+1$ membership queries $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)} \in \Re^D$ to the classifier. For the algorithm to be ϵ-optimal, these queries must constrain all consistent classifiers in the family $\hat{\mathcal{F}}^{\text{convex,'+'}}$ to have a common point

among their ϵ-$IMAC$ sets. Suppose that all the responses are consistent with the classifier f defined as

$$f(\mathbf{x}) = \begin{cases} +1\,, & \text{if } A_2(\mathbf{x}) < C_0^- \\ -1\,, & \text{otherwise} \end{cases}; \tag{C.9}$$

For this classifier, $\mathcal{X}_f^+$ is convex since A_2 is a convex function, $\mathbb{B}^{C_0^+}(A_2) \subset \mathcal{X}_f^+$ since $C_0^+ < C_0^-$, and $\mathbb{B}^{C_0^-}(A_2) \not\subset \mathcal{X}_f^+$ since $\mathcal{X}_f^+$ is the open C_0^--ball whereas $\mathbb{B}^{C_0^-}(A_2)$ is the closed C_0^--ball. Moreover, since $\mathcal{X}_f^+$ is the open C_0^--ball, $\nexists\ \mathbf{x} \in \mathcal{X}_f^-$ suchthat $A_2(\mathbf{x}) < C_0^-$ therefore $MAC(f, A_2) = C_0^-$, and any ϵ-optimal points $\mathbf{x}' \in \epsilon\text{-}IMAC^{(*)}(f, A_2)$ must satisfy $C_0^- \leq A_2(\mathbf{x}') \leq (1+\epsilon)C_0^-$.

Now consider an alternative classifier g that responds identically to f for $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$ but has a different convex positive set $\mathcal{X}_g^+$. Without loss of generality suppose the first $M \leq N$ queries are positive and the remaining are negative. Let $\mathbb{G} = conv\left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)}\right)$; that is, the convex hull of the M positive queries. I assume $\mathbf{x}^A \in \mathbb{G}$ since if it is not, then malicious classifier can construct the set $\mathcal{X}_g^+$ as in the proof for Theorems 6.5 and 6.4 above and achieve $MAC(f, A_2) = C_0^+$ thereby showing the desired result. Otherwise when $\mathbf{x}^A \in \mathbb{G}$, consider the points $\mathbf{z}^{(i)} = C_0^- \frac{\mathbf{x}^{(i)}}{A_2\left(\mathbf{x}^{(i)}\right)}$; *i.e.*, the projection of each of the positive queries onto the surface of the ℓ_2 ball $\mathbb{B}^{C_0^-}(A_2)$. Since each positive query lies along the line between $\mathbf{x}^A$ and its projection $\mathbf{z}^{(i)}$, by convexity and the fact that $\mathbf{x}^A \in \mathbb{G}$, the set $\mathbb{G}$ is a subset of $conv\left(\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(M)}\right)$. I refer to this enlarged hull as $\hat{\mathbb{G}}$. These M projected points $\left\{\mathbf{z}^{(i)}\right\}_{i=1}^M$ must form a covering of the C_0^--hypersphere as the loci of caps of half-angle $\phi^* = \arccos\left(\frac{1}{1+\epsilon}\right)$. If not, then there exists some point on the surface of this hypersphere that is at least an angle ϕ^* from all $\mathbf{z}^{(i)}$ points and the resulting ϕ^*-cap centered at this uncovered point is not in $\hat{\mathbb{G}}$ (since a cap is defined as the intersection of the hypersphere and a halfspace). Moreover, by definition of the ϕ^*-cap, it achieves a minimal ℓ_2 cost of $C_0^- \cos\phi^*$. Thus, if the adversary fails to achieve a ϕ^*-covering of the C_0^--hypersphere, the alternative classifier g has $MAC(g, A_2) < C_0^- \cos\phi^* = \frac{C_0^-}{1+\epsilon}$ and any $\mathbf{x} \in \epsilon\text{-}IMAC^{(*)}(g, A_2)$ must have

$$A_2(\mathbf{x}) \leq (1+\epsilon)MAC < (1+\epsilon)\frac{C_0^-}{1+\epsilon} = C_0^-$$

whereas any $\mathbf{y} \in \epsilon\text{-}IMAC^{(*)}(f, A)$ must have $A(\mathbf{y}) \geq C_0^-$. Thus, there are no common points in the ϵ-$IMAC^{(*)}$ sets of these consistent classifiers (*i.e.*, $\epsilon\text{-}IMAC^{(*)}(f, A) \cap \epsilon\text{-}IMAC^{(*)}(g, A) = \emptyset$) and so the adversary would have failed to ensure ϵ-multiplicative optimality. Thus, an ϕ^*-covering is necessary for ϵ-multiplicative optimality for ℓ_2 costs. However, from Lemma A.1, to achieve an ϕ^*-covering requires at least

$$M \geq \left(\frac{1}{\sin\phi^*}\right)^{D-2}$$

queries. Using the trigonometric identity $\sin(\arccos(x)) = \sqrt{1-x^2}$ and substituting for ϕ^* yields the following bound on the number of queries required for a given multiplicative

accuracy ϵ:

$$\begin{aligned} M &\geq \left(\frac{1}{\sin\left(\arccos\left(\frac{1}{1+\epsilon}\right)\right)} \right)^{D-2} \\ &\geq \left(\frac{(1+\epsilon)^2}{(1+\epsilon)^2 - 1} \right)^{\frac{D-2}{2}} . \end{aligned}$$

□