Secure Learning in Adversarial Environments

By

Bo Li

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

August, 2016

Nashville, Tennessee

Approved:

Yevgeniy Vorobeychik, Ph.D.

Bradley Malin, Ph.D.

Xenofon Koutsoukos, Ph.D.

Gautam Biswas, Ph.D.

Daniel Fabbri, Ph.D.

I dedicate this work to my parents.

ACKNOWLEDGMENTS

Pursuing a PhD and moving to a foreign country has been a valuable and exciting experience. I could not have succeeded without the advice, support and influence of many colleagues, friends and family. My thanks are due first and foremost to my advisor, Yevgeniy Vorobeychik, for his guidance and continuous support throughout my graduate career. Prof. Vorobeychik gives me consistently good advice and challenges, the opportunity to work with people with different expertise, and tremendous freedom to explore my research interests. His sharp sense for research and positive attitude will always been an inspiration to me. I will be forever grateful for his great helps!

I also want to thank my committee members, Professor Bradley Malin, Professor Xenofon Koutsoukos, Professor Gautam Biswas, and Professor Daniel Fabbri. They are always patient with my endless questions and taught me how to think and write in scientific ways. Without the fruitful discussions with them, I will not be able to come up with all these wonderful ideas and solve the tough problems.

I would like to express my appreciation to Symantec Research Labs for their graduate student fellowship financial support and research intern experience. Special thanks go to Kevin Roundy and Chris Gates who were my mentor during my internship for offering me numerous help on my research projects and advice me how to write and apply for patents.

Finally, I am grateful to my families for having supported me for my doctoral endeavor no matter what difficulties I face. I also would like to thank all my awesome lab mates and friends Sweta Panda, Jian Lou, Haifeng Zhang, Yonglong Ge, Ayan Mukhopadhyay, Liyiming Ke, and Royce Mou for the generous supports, helps and encouragements. They will all be my families forever!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Machine learning has become ubiquitous in the modern world, varying from enterprise applications to personal use cases and from image annotation and text recognition to speech captioning and machine translation. Its capabilities in inferring patterns from data have found great success in the domains of prediction and decision making, including in security sensitive applications, such as intrusion detection, virus detection, biometric identity recognition, and spam filtering. However, strengths of such learning systems of traditional machine learning are based on the distributional stationarity assumption, and can become their vulnerabilities when there are adversarial manipulations during the training process (poisoning attack) or the testing process (evasion attack). Considering the fact that the traditional learning strategies are potentially vulnerable to security faults, there is a need for machine learning techniques that are secure against sophisticated adversaries in order to fill the gap between the distributional stationarity assumption and deliberate adversarial manipulations. These techniques will be referred to as *secure learning* throughout this thesis.

To conduct systematic research for this *secure learning* problem, my study is based on three components. First, I model different kinds of attacks against the learning systems by evaluating the adversaries capabilities, goals and cost models. Second, I study the secure learning algorithms that counter any targeted malicious attacks by considering the specific goals of the learners and their resource and capability limitations theoretically. Concretely, I model the interactions between the defender (learning system) and attackers as different forms of games. Based on the game theoretic analysis, I evaluate the utilities and constraints for both participants, as well as optimize the secure learning system with respect to adversarial responses. Third, I design and implement practical algorithms to efficiently defend against multi-adversarial attack strategies.

My thesis focuses on examining and answering theoretical questions about the limits

of classifier evasion (evasion attack), adversarial contamination (poisoning attack) and privacy preserving problem in adversarial environments, as well as how to design practical resilient learning algorithms for a wide range of applications, including spam filters, malware detection, network intrusion detection, recommendation systems, etc. In my study, I tailor my approaches for building scalable machine learning systems, which are demanded by modern big data applications.

Chapter 1

INTRODUCTION

1.1 Motivation

The success of machine learning has led to its widespread use as an efficient tool in a wide variety of domains, from natural language processing, face detection, and handwriting recognition, to trading agent design [1, 2, 3]. As mentioned by Mitchell, learning approaches are particularly well-suited to domains where either the application is too complex to be designed manually, or needs to dynamically evolve. Many of the challenges faced in modern enterprise systems meet these criteria and stand to benefit from intelligent learning algorithms that are 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 [4, 5], enterprise-level network fault diagnosis [6, 7], and spam detection [8]. Machine learning has great utility when applied in security, networking, and large-scale systems as an intelligent tool for data analysis and autonomic decision making. It has also made significant inroads into security applications, such as fraud detection, computer intrusion detection, web search, and comparison shopping [9, 10, 11, 12]. For example, Google uses machine learning to identify websites engaged in Phishing [13]. Zozzle uses machine learning to identify malicious JavaScript programs [14], and Spam is typically identified using machine learning [15].

However, every coin has two sides. The strengths of machine learning approaches are their adaptability and ability to learn patterns that can be used for prediction or decision making. However, the learning systems can potentially be subverted by adversarial manipulations, which exposes applications that use machine learning techniques to a new class

of security vulnerabilities. For instance, the learning process can be susceptible to attacks that can cause the learner to disrupt the system it was intended to improve. With growing financial incentives of cyber-crime, more and more sophisticated adversaries are making efforts to conduct attacks against learners to disrupt the operations of or otherwise damage enterprise systems. 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. For example, spammers have regularly adapted their ideal 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 learning system.

The primary flaw in learning systems lies in the assumptions made about the learner's data that the training and test data comes from a natural or well-behaved distribution that remains stationary over time. However, such assumptions are perilous in the adversarial environments, where a patient adversary has a motive and the capability to alter the data used by the learner for training or prediction. Therefore, the learning systems can be compromised by an intelligent adversary for their own gains and makes the learning and adaptability properties of the original system into potential liabilities rather than benefits. In my research I analyze what are the strategies used by the adversaries, their utility and cost models, as well as the alternative defense methods that can bolster the system's resilience to an adversary. I consider several potential adversarial strategies posed to a learning system. One threat is that an attacker can exploit the adaptive nature of a machine learning system to modify the training process and cause it to fail. Here, the failures consists of causing the learning system to produce classification errors: if it misidentifies a hostile instance as benign (false negative), then the hostile instance is erroneously permitted through the security barrier; if it misidentifies a benign instance as hostile (false positive), then a permissible instance is erroneously rejected and normal user activity is interrupted. 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 and availability of the

system. By looking into what techniques a patient adversary can use to mislead or evade a learning system and how system designers would assess the vulnerability of their system to vigilantly incorporate trustworthy learning methods, it is possible to design robust learning algorithms that are resilient to the contaminated training process. The other threat is that an attacker can exploit the vulnerabilities of the learning system by probing or other offline analysis to modify the test data; therefore evade or cause the learning system to produce unreliable predictions. Of particular interest in my research work would be analyzing the adversarial strategies and designing corresponding robust learning algorithms, as well as applying them to evaluate the real-world systems.

Developing robust learning is not only of interest in its own right, but also especially important for security practitioners. To effectively apply machine learning as a general tool for reliable decision-making and prediction in computer systems, it is necessary to investigate how these learning systems perform under adversarial conditions. Without an in-depth understanding of the performance of these algorithms in an adversarial setting, the systems will not be trusted and may even cause serious financial loss [**?** **?**]. Therefore secure learning is of great importance in this big data era, the goal for which is to construct a learning algorithm that performs well under a realistic adversarial setting. Of course, whether an algorithm's performance is acceptable is a highly subjective judgment that depends both on the constraints placed on the adversary and on the job the algorithm is tasked with performing. Based on the fundamental concerns: the relevant security criteria to evaluate the security of a learner in a particular adversarial environment, and how to design or select the robust machine learning techniques satisfying the security requirements in a given problem domain, my research aims to design adversary aware learning algorithms that can take potential attacks into account during learning and try to provide systematic ways to evaluate the learning systems.

1.2 Research Challenges

Threat Model with Uncertainty and Diversity To proactively design the robust secure learning algorithms, one needs to identify potential vulnerabilities of machine learning algorithms during learning and classification; devise appropriate attacks that correspond to the identified threats and evaluate their impact on the targeted systems; and design countermeasures to improve the security of machine learning algorithms against the considered attacks. However, it is hard to predict and analyze the novel attacks appearing everyday based on their uncertain strategies and diverse malicious purposes. Although game theoretic approaches seem to be promising, understanding whether and to what extent the resulting attacker's strategies are well-representative of realistic and practical scenarios still remains an open issue [16, 17]. The main reason is that adversarial classification is not a game with well-defined rules such as board games (e.g. chess), and therefore, the real attacker's objective may not even correspond to a proper payoff function. It may therefore be useful to verify, on the contrary, in a reactive manner. So knowledge about whether real attackers behave as hypothesized, and how to build threat models from the observed manipulations on real attacks is important. Another issue is that most formulations, such as Nash and repeated games, do not account for limited knowledge of the attacker's objective function explicitly, which may reflect in turn the limited attacker's knowledge about the classifier. Even when the limited knowledge is taken into account, such as the Bayesian game [18], the uncertainty on the payoff function is simply modeled with a prior distribution over its parameters, therefore the payoff function is still essentially know. Therefore, how to design the secure learning algorithm which can consider the uncertainty and diversity of the real world adversarial models, as well as reflect the limited knowledges of both the learning system and adversaries to avoid over-estimation therefore retaining system accuracy remains major challenge.

Unlimited Repetition Considering the nature of sophisticated adversaries, the interactions between the learning systems and the adversaries can form a repeated game which

will never end. Therefore, how to terminate the optimization process and make decisions based on the designed stopping criteria, such as the adversarial cost estimation and tradeoff between accuracy and robustness for the learning system, form an obstacle for the secure learning problem.

Scalability Another challenge for the secure learning development is the scalability of the training procedure on large datasets. To capture the diverse adversarial strategies, the robust learning process with invariance is essentially a minmax approach in which the learning algorithm is modified to minimize the maximum loss based on the worst-case manipulations of the attack samples [19]. However, solving such minmax problems with large datasets based on a large number of constraints, each of which models a kind of adversarial strategy, is computationally challenging and requires advanced scalable or parallel computation techniques.

Real World Data It is difficult to collect the updated real attacked datasets for the adversarial learning. For instance, the spam data or network intrusion data published nowadays is either out of date or failing to contain real world attacks. Therefore, the simulated attacks should be carefully designed and the adversarial estimation error should also be injected to ensure the robustness of the designed algorithm.

Proactive vs. Reactive Strategies Based on the challenges mentioned above, a thorough proactive security evaluation, considering all possible attacks, may be infeasible, and designing the corresponding countermeasures may be even more difficult [20, 21]. Therefore, how to combine the proactive with reactive defense strategies together to build the robust learning system by forecasting the novel attacks as well as updating the defense methods based on the adversarial feedback form another challenge.

Balance between Security/Privacy vs. Utility One nature strategy to enhance security is to inject randomness to form the non-deterministic system, and therefore decrease the probability of adversarial success. In cryptography, good random numbers are fundamental to almost all secure computer systems. Without them, everything from Second World

War ciphers like Lorenz to the SSL the browser uses to secure web traffic are in serious trouble. Similarly, in data privacy protection system, for example, assigning random ID to users is often applied to preserve personal information. However, such randomness can also harm the utility of learning systems by decrease their prediction efficiency and accuracy. Thus, how to control the amount of randomness injected and how to select proper randomness with respect to specific tasks to minimize the system utility loss lead to another big challenge.

1.3 Overview of Thesis

The remainder of this thesis is organized as below. In the following Chapter 2, I present the motivations, background knowledge and related materials for *secure learning*. Chapter 3- 7 focus on the adversary model analysis and provide robust defensive strategies to demonstrate the feasibility and value of the secure learning systems against evasion attacks from different perspectives. I first show and study a real world problem, singleton malware detection, in Chapter 3. Then I provide theoretical analysis as well as derive the robust learning algorithms considering various attack strategies by modeling the interaction between learner and attacker as games in Chapter 4 and 8. Besides the mathematical modeling strategy, I then present the human subject based study for deriving attack models from the real collected adversarial data in Chapter 6. To generalize the robust learning strategy against evasion attacks, Chapter 7 provides an efficient robust learning framework to take different adversary strategies into account based on any learning algorithms. Chapter 9 starts to study poisoning attacks. In this chapter I take the recommendation system as an example to analyze how the adversaries derive poisoning instances to contaminate the training data while still mimicking the normal users' behavior patterns. Finally, I study the application of *secure learning* on privacy settings in Chapter 10. This chapter focuses on the theoretic analysis for attack models in privacy preserving data publishing model. I provide a iterative data sanitization algorithm to balance the data utility and privacy to protect

data privacy. Conclusions and future work for this thesis are then discussed in Chapter 11.

Part I Robust learning against evasion attacks

Chapter 2

A REAL WORLD CASE: LARGE-SCALE IDENTIFICATION OF MALICIOUS SINGLETON FILES

The study of this chapter is based on a real-world dataset of billions of program binary files that appeared on 100 million computers over the course of 12 months, discovering that 94% of these files were present on a single machine. Motivated by this practical malware detection problem, we looked into the secure learning problem by taking the adversarial strategies into account. Though malware polymorphism is one cause for such large number of singleton files, additional factors also contribute to polymorphism, given that the ratio of benign to malicious singleton files is 80:1. The huge number of benign singletons makes it challenging to reliably identify the minority of malicious singletons. Here we present a large-scale study of the properties, characteristics, and distribution of benign and malicious singleton files. We leverage the insights from this study to build a classifier based purely on static features to identify 92% of the remaining malicious singletons at a 1.4% percent false positive rate, despite heavy use of obfuscation and packing techniques by most malicious singleton files that we make no attempt to de-obfuscate. Finally, we demonstrate robustness of our classifier to important classes of automated evasion attacks and emphasize the necessity for *secure learning*.

2.1 Overview

Despite continual evolution in the attacks used by malicious actors, labeling software files as benign or malicious remains a key computer security task, with nearly 1 million malicious files being detected per day [88]. Modern anti-virus vendors deploy a suite of techniques that together label software as benign or malicious, each analyzing the software

from a different perspective. Some of the most reliable techniques label files by combining the context provided by multiple instances of the file. For example, Polonium judges a file based on the hygiene of the machines on which it appears [89], while Aesop labels a file by inferring its software-package relationships to known good or bad files, based on file co-occurrence data [90]. These detection technologies are unable to protect customers from early instances of a file because they require the context from multiple instances to label malware reliably, only protecting customers from later instances of the file. Thus, the hardest instance of a malware file to label is its first, and regrettably, the first instance is also the last in most cases, as most malware samples appear on a single machine. In 2015 around 89% of all program binary files (such as executable files with .EXE and .DLL extensions on Windows computers) reported through Norton's Community Watch program existed on only one machine, a rate that has increased from 81% since 2012. To make matters worse, real-time protection must label files that have been seen only once even though they may eventually appear on many other machines, putting the effective percentage of unique files at any given time at 94%.

I present the first large-scale study of *singleton* files and identify novel techniques to label such files as benign or malicious based on their contents and context. We define a singleton file as any file that appears on exactly 1 machine. I consider two files to be distinct when a cryptographic hash taken over their contents (such as SHA-256) yields a different result, meaning that two files that differ by a single bit are considered distinct even though they may be functionally equivalent.

Due to the fact that malware is often polymorphic, many malicious files are among these singletons. However, singleton executable files do not trend towards being malicious; in fact the opposite is true: the ratio of benign to malicious singleton files is 80 to 1, resulting in a skewed dataset. This ratio gives low prevalence malware a large set of files to hide amongst and it makes effective classification models difficult to train, as most machine learning models require relatively balanced data sets for effective training. We study the

root causes behind the large numbers of benign singleton files in Section 3.3.2 and study malicious singletons in Section 3.3.3. We study the properties of machines that are prolific sources of benign singleton files in Section 3.4.1. I filter obviously benign singletons by profiling the prominent categories of benign singleton files that appear on such systems (Section 3.4.2). I present the full machine learning pipeline and the features I use to classify these samples in Section 3.4.3. I present experimental results in Section 3.5.

Since the phenomenon of malicious singleton files was largely driven by the arms race between security vendors and malicious adversaries in the first place, it is important that we analyze robustness of our model against evasion attacks, and I do so in Section 3.5.3. We form the interactions between and adversary and our malware detection system as a Stackelberg game [91] and simulate evasion attacks on real singleton files to demonstrate that our proposed pipeline performs robustly against attacker interference.

Figure 3.1 illustrates the malicious target in the learning framework, where attackers try to generate the data distribution closing to the original training data while satisfying the preferred malicious properties.

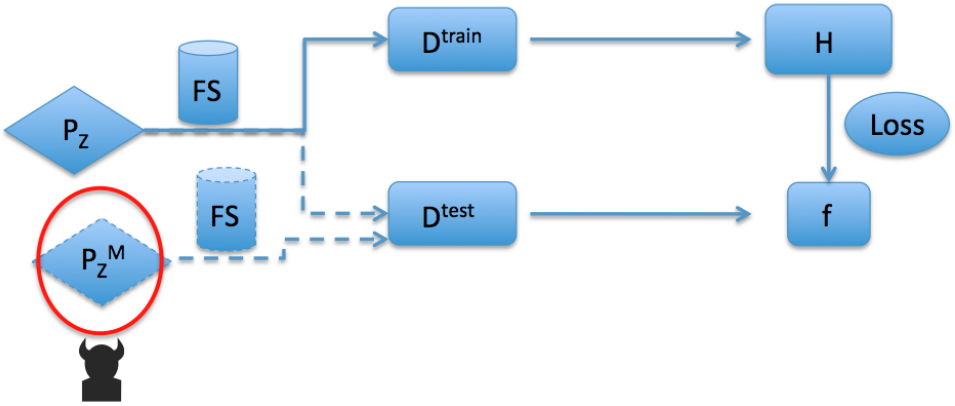


Figure 2.1: Illustration for polymorphic singleton malware attack strategy

2.2 Related work

The problem of detecting malicious files has been studied extensively. Particularly relevant is work that is designed to deal with low-prevalence malware. This prior art includes work designed to reverse the effect of packing-based obfuscation tools by either statically decompressing or decrypting the malicious payload [92], or simply executing the program until it has unrolled its malicious payload into main memory [93]. At this point, traditional anti-virus signatures may be applied [94], and clustering may serve to identify new malicious samples based on their similarity to known malicious samples [95, 96]. By contrast, we make no effort to undo obfuscation attempts, which are frequently evidence of malicious intent. Whereas these researchers have focused on the causes behind low-prevalence malware, we augment this by providing the first detailed study of benign singleton files. We leverage our study of the distribution, causes, and naming conventions of singleton files to distinguish between benign and malicious instances, adding a broad collection of static properties of these files to buttress our approach against adversarial mimicry attacks.

Researchers have also dealt with obfuscation by ignoring the program binary file altogether and instead classifying the malware based on dynamic traces of file and process [97] or network-based activity [98]. Dynamic approaches are typically used a last resort when static analysis attempts such as ours are unsuccessful, as they typically have the drawbacks of either allowing the machine to be infected before a post-mortem detection can be effected, or must be run at considerable cost in an instrumented sandbox, in which the suspected file's behavior on the targeted machine can be difficult to replicate.

The importance of an adversarially robust approach to malicious singleton detection is evident, given that the high volume of singleton malware is largely the byproduct of adaptations to anti-virus technology. Researchers have formalized the notion of evasion attacks on classifiers through game theoretic modeling and analysis [44, 99]. In one of the earliest such efforts, Dalve et al. [44] play out the first two steps of best response dynamics in this game. Bruckner et al. [100] examine single-shot prediction games, where the utility func-

tions of learner and adversary are not necessarily antagonistic, and propose algorithms to find the equilibrium. Later an alternative Stackelberg game model is suggested, in which the learner (leader) first sets the algorithm parameters, and the attacker (follower) would best respond by optimizing its utility [84]. In all of the prior work, there has been a disconnect between the learner-attacker game models and real world dataset validation. We bridge this gap by considering a very general adversarial learning framework based on an evaluation of a real, large-scale dataset.

2.3 Singleton Files in the Wild

To address the paucity of information about singleton files, I focus on their causes, distribution patterns, and internal structure. We describe the predominant reasons for which software creators produce benign and malicious singleton files. For benign singletons, we identify the software packages that are the strongest predictors of the presence of benign singleton files on a machine. For malicious software, many singletons are produced from a relatively much smaller base of malware families. Thus, to better understand the nature of the polymorphism that is present in practice across a large body of singleton malware, we study the static properties of malicious singleton files across all malware families and within individual families.

2.3.1 Dataset Description

In the interest of performing a reproducible study, we perform the following study over data that is voluntarily contributed by members of the Norton Community Watch program, which consists of metadata about the binary files on their computers and an identifier of the machines on which they appear. Symantec shares a representative portion of this anonymized dataset with external researchers through its WINE program [101]. We use an extended window of time from 2012 through 2015 to generate high-level statistics about singleton data, and refer to this dataset as $D0$. We also use an 8-month window of data from

2014 for a more in depth analysis of the properties of singleton files and machines on which they appear, we call this $D1$. A portion of the files in $D1$ is labeled with high-confidence benign or malicious labels. We form dataset $D2$ by selecting a subset of the previous data that consists of labeled singleton files, and for which the file itself is available, allowing us to extract additional static features from the files that we describe in Section 3.4.3. This dataset comprises 200,000 malicious and 16 million benign singleton files, and is the basis of the experimental evaluation of Section 3.5.

2.3.2 Benign Singleton Files

The abundance of benign singletons may be surprising given that there are not obvious benefits to distributing legitimate software as singleton files. Of course, some software is rare simply because it attracts few users, as in the case of software on a build machine that performs daily regression tests. However, there are also less obvious, but no less significant reasons behind the large numbers of singleton files, including the following:

1. The .NET Framework seeks to enable localized performance optimizations by distributing software in Microsoft Intermediate Language code so it can be compiled into native executable code by the .NET framework on the machine where it will execute, in a way that is specific to the machine's architecture. This is evident in practice, as .NET produces executables that are unique in most cases. Its widespread use makes it the largest driver of benign singleton files in our data.
2. Many classes of binary rewriting tools take a program binary file as input, producing a modified version as output, typically to insert additional functionality. For instance, tools such as Themida and Armadillo add resistance to tampering and reverse engineering, frequently to protect intellectual property and preserve revenue streams, as in the example of freemium games that require payment to unlock in-game features and virtual currency. Other examples of binary rewriting tools include the RunAsAdmin tool referenced in Table 3.1, which modifies executables so that administrative privileges are

required to run them.

3. In many cases, software embeds product serial numbers or license keys in its files, resulting in a different hash-based identifier for otherwise identical files.
4. Singleton files can be generated by software that produces executable files in scenarios where other file formats are more typically used. For instance, Microsoft's Active Server Pages framework generates at least one DLL for every ASP webpage that references .NET code. Another example is ActiveCode's Building Information Modeling software that creates project files as executables rather than as data files. It is not uncommon for these frameworks to generate thousands of singleton binaries on a single machine.
5. Interrupted or partial downloads can result in files that appear to be singletons, even though they are really prefixes of a larger more complete file. If the entire file is available for inspection, this can be checked, but our dataset includes metadata for many files that have not been physically collected.

In Figure 3.2 we show the most common substring used in benign singleton filenames as extracted from dataset $D1$, many of which hint at the above factors. In particular, the most-observed filename pattern is "app-web-", which is seen in DLL files supporting webpages created by ASP Web Applications. These files are often singletons because they are compiled from .NET code.

Using a subset of the data from dataset $D0$, we demonstrated in Figure 3.3 that singleton files are not uniformly distributed across systems. The figure shows the number of machines that possess specific counts of singleton and non-singleton files. Figure 3.4 is another way to view the same data, showing that almost 40% of machines have few or no singleton files and more than 94% of the systems have fewer than 100 singletons. Thus, the majority of singleton files come from the heavy tail of the distribution representing relatively few systems. Note that this data is from a specific period in time, and so machines with low numbers of non-singleton files indicate machines that experienced minimal changes/updates during the period when data was collected.

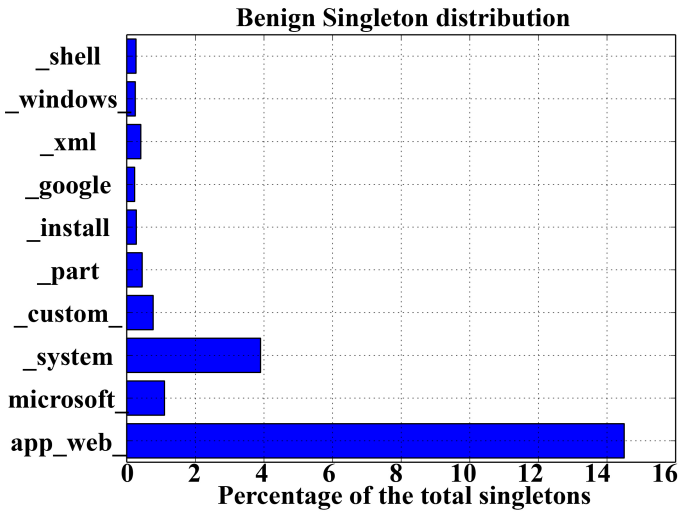


Figure 2.2: Percent of singleton files containing a specific substring.

To help us work towards a solution that could identify benign singletons as such, we seek to better understand the machines on which they are most likely to exist. To this end, we used dataset $D1$ to identify the software packages that are most indicative of the presence or absence of benign singletons on a computer. To identify software packages that could be responsible for the creation of singletons, we turn to the clustering approach proposed by Tamersoy et al. [90], which identifies software packages indirectly by clustering software files that are nearly always installed together on a machine, or not at all (see Section 3.4.1 for more details). Henceforth, we refer to these clusters as *software packages*. Once files are so clustered, we proceed by identifying the software packages that are most indicative of the presence of absence of singletons on a machine. Let S denote a specific software package (cluster). We identified a set of 10 million machines from $D1$, each of which contains at least 10 benign singleton files, which we denote by H (for *HasSingletons*). Likewise, we identified 10 million machines from $D1$ with no singleton files, which we denote by N, for *NoSingletons*. We identify the predictiveness of each software package S by counting its number of occurrences in each H and N, and use these counts to compute the odds ratios (OR) of a machine containing singletons given S, $OR(S) = H/N$. Intuitively, the higher $OR(S)$ is for a particular software package S, the more likely it is

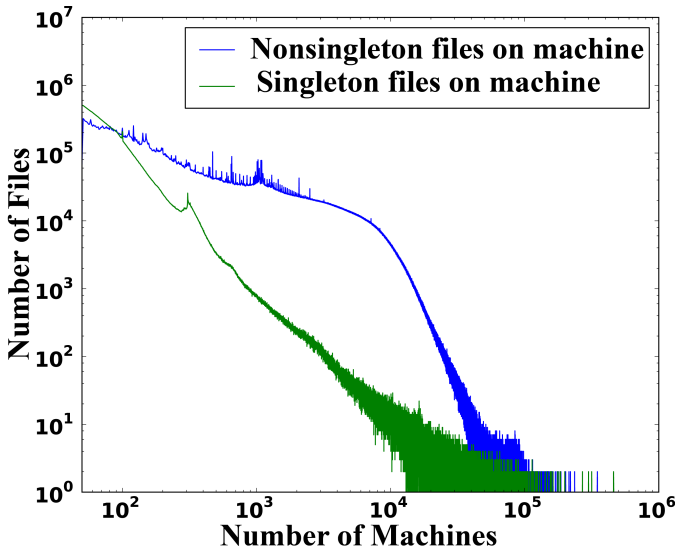


Figure 2.3: Number of machines with a specific number of singleton/non-singleton files. Based on data sampled from $D0$

Have singleton: Control set OR	Representative Filename	Software Name
13770:1	Appvux.dll	Microsoft App-V
11792:1	Soapsuds.ni.exe	SoapSuds Tool for XML Web Services
110501:2	Blpsmarthost.exe	SmartClient
36515:2	gtpo3d host.dll	Google Talk Plugin
13868:1	Runasadmin.exe	Microsoft RunAsAdmin Tool
8511:1	Microsoft.office.tools.ni.dll	Visual Studio
...	...	...
1:1702	Policy.exe	???
1:4392	vdiagentmonitor.exe	Citrix VDI-in-a-Box

Table 2.1: Software packages that are most predictive of presence/absense of benign singleton files. For succinctness, we represent each software package by its most prevalent filename.

that this (benign) package generates many singletons. An $OR(S)$ ratio that is close to 1 is indicative of a software package that is equally likely to appear on machines that do and do not contain singletons, and therefore probably does not generate singletons itself. On the other hand, an $OR(S)$ that is significantly lower than 1 indicates that machines on which S is installed are tightly controlled or special-use systems unlikely to contain singleton files.

Table 3.1 shows software packages that are strong predictors for the presence (or absence) of benign singletons on a machine. Software packages that correlate with increased numbers of singletons include compiler-related tools (Visual Studio, SoapSuds, SmartClient), tools that wrap or modify executables (RunAsAdmin, App-V), and software pack-

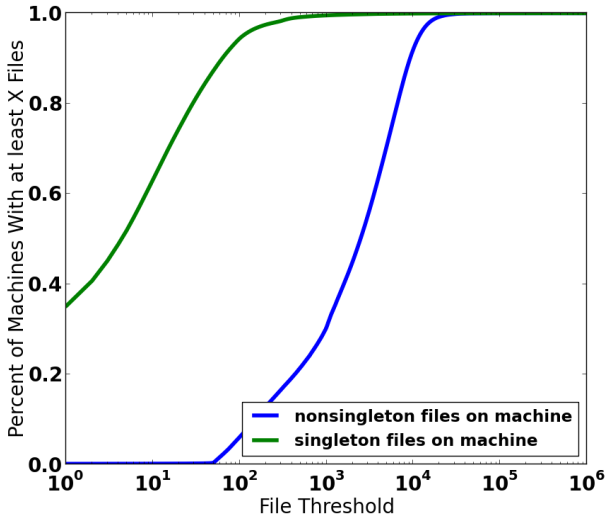


Figure 2.4: Percent of machines that report more than X singleton and non-singleton files. Based on data sampled from *D*0.

ages that include numerous signed singletons (Google Talk Plugin). Interestingly, there are also many software packages that correlate strongly with an absence of singletons on the system. These are indicative of tightly controlled or minimalist special-purpose systems.

Our ability to identify software packages that lead to presence/absence of many benign singleton files is a critical step towards developing a method for classifying malicious vs. benign singletons. In particular, as described in Section 3.4, it enables us to prune a large fraction of files as benign before applying machine learning methods, dramatically reducing the false positive rate.

2.3.3 Malicious Singleton Files

Malware files skew heavily towards low-prevalence files, and towards singleton files in particular. Using *D*0 we can see that this trend has increased in recent years: 75% of known malware files were singletons in 2012, and the rate increased to 86% by 2015. There are readily apparent reasons why malware files skew towards low-prevalence files, including the following:

1. Avoiding signature-based detection: Users typically want to prevent malware from running on their systems, and blocking a single high-prevalence file is much easier than blocking large numbers of distinct yet functionally equivalent files. Polymorphism is a widespread technique for producing many functionally equivalent program binaries, which aims to reduce the effectiveness of traditional Anti-Virus signatures over portions of the file.
2. Resistance to reverse engineering and tampering: Many malware authors pack, obfuscate or encrypt their binaries, often with the assistance of third-party tools that are inexpensive or free. Polymorphism is often a welcome byproduct of these techniques, though it is not necessarily the primary objective.
3. Malware attribution resistance: The ease with which malware authors can create many functionally equivalent malware files makes the problem of attributing a malicious file to its author much harder than it would be if the same file was used in all instances. For the same reason, polymorphism makes it difficult for security researchers to assess a malware family's reach. Modularity also allows for specific components to be used as needed, without unnecessarily exposing the binary to detection.

Despite the widespread availability and use of tools that can inexpensively apply polymorphism and obfuscation to malware binaries, the security industry has developed effective techniques to counter these. Much of the polymorphism seen in malware binaries is superficially applied by post-compilation binary obfuscation tools that "pack" the original contents of the malware file (by compressing or encrypting the code), and add layers of additional obfuscation-related code [102]. There are some obfuscation tools that are far more complex than this, but most of them are used almost exclusively by either malicious or by benign software authors. Techniques used by the anti-virus industry to combat these obfuscations are discussed at the end of this section.

To provide additional insight into the nature of malware polymorphism, we study the use of polymorphism by 800 malware families that were observed in the wild in our $D1$

dataset. Overall, we found that 31% of these families are distributed exclusively as singletons, accounting for over 80% of all singleton malware files, while 60% of families rely exclusively on non-singletons. There is a subtle difference here, that by volume, the 60% of families account for many detections since they are higher prevalence, while the 80% of singletons account for a lower percent of all detections even though there are more of them, since they only occur on a single system.

To identify malware families that exhibit a high degree of polymorphism, we extracted about 200 static features from files belonging each malware family. Our features include most fields in the Portable Executable file header of Windows Executable files (such as file size, number of sections, etc.), as well as entropy statistics taken from individual binary sections, and information about dynamically linked external libraries and functions that are listed in the file's Import Table. For each malware family, we calculate variability scores as the average variance of our static features for the files belonging to that family. The families with the highest variability scores are:

- *Adware.Bookedspace*
- *Backdoor.Pcclient*
- *Spyware.EmailSpy*
- *Trojan.Usuge!gen3*
- *W32.Neshuta*
- *W32.Pilleuz*
- *W32.Svich*
- *W32.Tu1ik*

These malware families vary greatly in form, function, and scale, though they do share properties that help account for their high variance. In particular, all of these families are modular, infecting machines with multiple functionally different files that are of similar prevalence and have dramatically different characteristics. In all cases, there is at least an order of magnitude difference in file size between the largest and smallest binary. Further-

more, all samples apply binary packing techniques sporadically rather than in all instances.

Backdoor.Pcclient is a Remote Access Trojan and the lowest prevalence family that has high variance in the static features. Polymorphism is not evident in this family; its elevated variance is a reflection of a modular design, multiple releases of some of those modules, and large differences from one module to another. By contrast, *W32.Pilleuz* is a very prevalent worm family, but its Visual Basic executables achieve high variance through extensive obfuscation and highly variable file sizes, which add to the worm's modularity and occasional use of packing. *W32.Neshuta* is particularly interesting in that it infects all *.exe* and *.com* files on the machines that it compromises, resulting in many detected unique executables of differing sizes, in addition to its own modular and polymorphic code.

API Purpose	*API Function*
Anti-Analysis	*IsDebuggerPresent* *GetCommandLineW* *GetCurrentProcessId* *GetTickCount* *Sleep* *TerminateProcess*
Unpack Malware Payload	*GetProcAddress* *GetModuleHandleW* *GetModuleFileNameW*
Load/Modify Library Code	*CreateFileMappingA* *CreateFileMappingW* *MapViewOfFile* *SetFilePointer* *LockResource*
Propagation	*GetTempPathW* *CopyFileA* *CreateFileW* *WriteFile*

Table 2.2: Categories of Windows API functions that are disproportionately used by malware

The Windows API functions imported by malware files provide interesting insights into their behavior, and are useful as static features, because they are reasonably adversarially resistant. Though malware authors can easily add imports for API functions that they do

not need, removing APIs is significantly harder, as these may be needed to compromise the system (e.g., CreateRemoteThread). The only inexpensive way in which a malware file can hide its use of API functions from static analysis is to use a binary packing tool so that its Import Table is not unpacked until runtime, when it is used to dynamically link to Windows API functions. However, this technique completely alters the file's static profile and introduces the static fingerprint of the obfuscation tool, offering an indication that the file is probably malicious. In addition, as discussed at the end of this section, these obfuscations can be reversed by anti-virus vendors.

Table 3.2 lists the API functions that are most disproportionately used by malware, categorized by the purpose for which malware authors typically use them. Many of these APIs support analysis resistance, either by detecting an analysis environment, hiding behavior from analysis, or by actively resist against analysis. Most other APIs that are indicative of malware have to do with linking or loading to additional code at runtime, typically because the malware payload is packed, but also for more nefarious purposes, such as malicious code injection and propagation.

Anti-Virus Industry Response to Obfuscation

The anti-virus industry has sought to adapt to malware's widespread use of obfuscation tools by applying static and dynamic techniques to largely reverse the packing process in a way that preserves many of the benefits of static analysis. In particular, these techniques allow malicious code to be extracted, along with the contents of the Import Address Table, which contains the addresses of functions imported from external dynamically linked libraries. Unpacking techniques include the "X-Ray" technique, which may be used to crack weak encryption schemes or recognize the use of a particular compression algorithm and reverse its effects [92]. Most unpacking techniques, however, have a dynamic component and can be broadly classified into emulators and secure sandboxes. Emulators do not allow the malicious files to execute natively on the machine or to execute real system calls or

Windows API calls, but provide a reasonably good approximation of a native environment nonetheless. They are frequently deployed on client machines so that any suspicious file can be emulated long enough to allow unpacking to occur, after which the program's malicious payload can be extracted from memory and the de-obfuscated code can be recovered and analyzed. Offline analysis of suspicious program binaries typically uses a near-native instrumented environment where the malware program can be executed and its dynamically unpacked malicious payload can be extracted [93]. Though there are more elaborate obfuscation schemes that can make executable files difficult to unpack with the aforementioned techniques, these are either not widely deployed (e.g., because they are custom-built for the malware family) or are used predominantly by benign or malicious software, but not both. Thus, effective benign vs. malicious determinations can be made even in these cases, because the obfuscation toolkits leave a recognizable fingerprint.

Though the effectiveness of the above de-obfuscation techniques is open to debate, in our methodology here, we make the deliberate choice to use no de-obfuscation techniques at all in our attempts to classify singleton files. We demonstrate that malware classification based purely on static features can be successful, even in the face of extensive polymorphism, by good and bad files alike. The success we achieve demonstrates that the obfuscation techniques that are widely used by malware are themselves recognizable, and appreciably different from the kinds of polymorphism that are common in benign files. We expect that the classification accuracy of our methodology would improve when applied to files that have been de-obfuscated, given that other researchers have found this to be the case [95].

2.4 Learning for Labeling Singletons

Most prior efforts for identifying malicious files have either relied on the context in which multiple instances of the file appear (e.g., Polonium [89] and Aesop [90] systems) or have relied exclusively on static or dynamic features derived from the file itself (e.g.,

MutantX-S [95]). The context that is available for a singleton file is necessarily limited, making the aforementioned context-dependent techniques not applicable. Making matters worse is the fact that the ratio of benign to malicious singleton files is nearly 80:1, which has the effect of multiplying the false positive rate of a malware detector by a factor of 80, and presents a significant class imbalance problem that makes effective classifier training difficult.

To address the lack of context for singleton files and the preponderance of benign singleton files, we leverage insights gleaned from our empirical observations about singleton files in the wild. In particular, as discussed in Section 3.3.2, a handful of software packages generate the lion's share of benign singletons, while other packages correlate with their absence. Furthermore, the toolchains that generate benign singletons in large numbers imbue them with distinctive static properties that make them easy to label with high confidence. We use these insights to develop a pipeline that filters benign singleton files with high confidence, yielding a more balanced dataset of suspicious files. We extract static features from the remaining suspect files and apply supervised learning to classify them as benign or malicious.

Figure 3.5 presents a diagram of our pipeline. We take as input a pair (f, m), where f is a file and m is the machine on which it resides. The first step of the pipeline, which we call *machine profiling*, determines whether m is likely to host many benign singleton files. The second step is *file profiling*, in which we label obviously benign files, primarily from many-singleton machines, by determining that they closely match the benign files that are common on such systems. The final step, *classification*, uses a supervised classification algorithm (we explore the use of Support Vector Machines [103] and Recursive Neural Networks [104]) to render a final judgment on the remaining files. We proceed by describing each of our pipeline's components in detail.

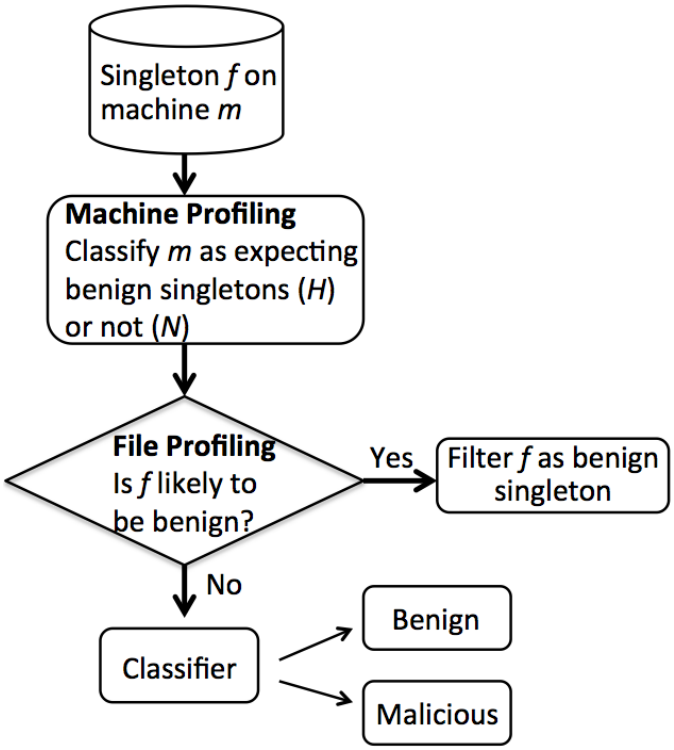


Figure 2.5: Pipeline of the singleton classification system.

2.4.1 Machine profiling

Machine profiling operationalizes the following insight gleaned from our empirical observations: since the distribution of benign singletons is highly non-uniform, singleton classification will benefit from identifying machines that are likely to host many benign singletons. As discussed in Section 3.3.2, the software packages present on a machine are highly predictive of the presence or absence of benign singletons.

The first challenge we face is that of automatically identifying software packages from telemetry about installations of individual program binary files. In mapping individual files to software packages, we wish to proceed in an automated way that is inclusive of rare software that is not available to be publicly downloaded. Our approach adopts the clustering portion of the Aesop system described by Tamersoy et al. [90], in which they leverage a dataset consisting of tuples of file and machine identifiers, each of which indicates the presence of file f on machine m. Specifically, let F be a set of (high-prevalence) files (in the

training data). For each file $f \in F$, let $M(f)$ be the set of machines on which f appears. As Aesop did, we use locality sensitive hashing [105] to efficiently and approximately group files whose $M(f)$ sets display low `Jaccard` distance to one another. The Jaccard distance between two sets X and Y is defined as: $J(X,Y) = 1 - \frac{X \cap Y}{X \cup Y}$, and we define the distance between two files f and f' in terms of Jaccard distance as $d(f,f') = J(M(f),M(f'))$. We tune locality sensitive hashing to cluster files with high probability when the Jaccard distance between the files is less than 0.2, and to cluster them very rarely otherwise. We obtain a collection of clusters $\mathscr{C}$, such that each cluster $C \in \mathscr{C}$, serves as an approximation of a software package, since C represents a collection of files that are usually installed together on a machine or not at all.

We proceed by identifying the approximate software packages that are the best predictors for the presence of singleton files. We formulate this task as a machine learning problem. We define a feature vector for each machine m that encodes the set of software packages that exist on m. Specifically, given n clusters (software packages), we create a corresponding binary feature vector s_m of length n, where $s_{mj} = 1$ iff cluster j is present on machine m. Next, we append a label l_m to our feature vector such that we have $\{s_m, l_m\}$ for each machine, with feature vectors s_m corresponding to machines and labels $l_m \in \{H,N\}$ representing whether the associated machine ***h***as benign singletons (label H) or has ***n***o singletons (label N). With this dataset in hand, we are able to train a simple, interpretable classifier to predict l_m to good effect. Had we used individual files as predictors, we would have to choose a machine learning algorithm that behaves well in the presence of strongly correlated features, but software package identification dramatically reduces feature correlation. Thus, we select Naive Bayes as our classifier $g(s)$, which performs well and gives us significant insight into the software packages that are the best indicators of the presence or absence of benign singleton files, as reported in Table 3.1. Our classifier takes as input a feature vector s that represents the software packages on a given machine, and outputs a prediction as to whether or not the machine has benign singletons. To achieve a bal-

anced dataset, we randomly selected 2,000,000 uninfected machines, half of which contain singletons and half of which do not.

2.4.2 File profiling

Given a classifier $g(m)$ that determines whether a machine m is expected to host benign many singletons, the next step in our pipeline—*file profiling*—uses this information to identify files that can be confidently labeled benign. The result is both a more balanced dataset that makes our pipeline's classifier easier to train, as well as a high-confidence labeling technique that reduces classifier's false positive rate. The main intuition behind our proposed file profiling method is that benign singleton files bear the marks of the specific benign software packages that generate them. Of course, different software generates singletons with dramatically different file structures and file-naming conventions. Consequently, we seek to identify prototypical benign singletons by clustering them based on their static properties, and filter benign files that closely match these prototypes. Since the information we have about the software installed on any given machine is typically incomplete, we filter benign files that closely match benign-file prototypes on all machines, but require much closer matches on machines where benign singletons are not expected. This point is operationalized below through the use of a less aggressive filtering threshold for machines m labeled as N (no benign singletons) than for machines labeled H (having benign singletons).

The full path, filename, and size of singleton files are the primary static attributes that we use in our file profiling study. We had little choice in this case because large collections of labeled benign singleton files that security companies share with external researchers are extremely hard to come by, and are limited in the telemetry they provide. In the interest of conducting a reproducible experiment, we limit ourselves to the metadata attributes provided for files in Dataset $D1$ (see Section 3.3.1) that Symantec shares with external researchers through its WINE program [101]. Though $D1$ gives us a representative dataset

of singleton files, it also limits us to a small collection of metadata attributes about files, of which the path, filename, and size are the most useful attributes. In modest defense of the use of filename and path as a feature, though it is true that a malicious adversary can trivially modify the malware's filename (and the path, to a lesser extent), the malware author would frequently have to do so at the cost of losing the social engineering benefit of choosing a filename that entices the user to install the malware. We note that security vendors could readily remediate the limitations of our file profiling study by adding more adversarially resistant static features to a file profiling model, as they have the flexibility to collect additional telemetry for singleton files.

We proceed by developing techniques to maximize the discriminative value of the path and filename. We seek to leverage the observation that a handful of root causes create a significant majority of benign singletons, and these origins are often strongly evident in the filename and path of benign singletons. Although malware files display significantly more diversity in their choice of filenames, these filenames typically bear the marks of social engineering, and their paths are frequently reflective of the vector by which they managed to install themselves on the machine, or are demonstrative of attempts to hide from scrutiny. Accordingly, we engineer features from filenames and paths such that they are capable of capturing the naming conventions used by benign singletons. Given a file f, we divide its filename into words using chunking techniques. Specifically, we identify separate words within each file name that are demarcated by whitespace or punctuation, and separate words based on CamelCase capitalization transitions, and so on. Subsequently, we represent the filename and path components in a "bag of words" feature representation that is physically represented as a binary vector, where the existence of a word in the filename or path corresponds to a 1 in the associated feature, and a 0 indicates that the word is not a part of its name. In addition, we capture the relative frequencies of the words that appear in filenames by measuring the term frequency (TF) of each word. Term frequency is then used as a part of weighted Jaccard distance measure used to cluster files,

as described below. More formally, let $T \subseteq \mathbb{R}^n$ represent the feature space of the singleton files, with n the number of features. Each singleton file f can be represented by a feature vector t, which is the dot product of a binary bag of words vector w and the normalized term frequency vector q corresponding to each word, $t = w \cdot q$, where t^j is the jth feature value. Note that we exclude words that appear extremely frequently, such as *exe, dll, setup*, as stop words, to prevent the feature vector t from becoming dominated by these. For any two files f_1 and f_2, the weighted Jaccard distance between them is then calculated as $J(f_1, f_2) = 1 - \frac{\sum_k \min({f_1}^k, {f_2}^k)}{\sum_k \max({f_1}^k, {f_2}^k)}$.

We use the weighted Jaccard distance to cluster benign singleton files in the training data using the scalable NN Descent algorithm [106] implemented on Spark [107], which efficiently approximates K-Nearest Neighbors and produces clusters C of of highly similar files.[1] We gain further efficiency and efficacy gains by choosing a bag of words representation over edit distance when making filename comparisons. This approach also has the benefit of producing an understandable model that identifies the most frequent filename patterns present in benign singleton files, such as those highlighted in Figure 3.2.

The final step in the file profiling process is to use the clusters derived above to filter benign files that align closely with the profile of benign singletons. To this end, for each benign singleton cluster $c \in C$, we compute the cluster mean $\bar{c} = \frac{1}{|c|} \sum_{t_j \in c} t_j$. For a given file f, we then find the cluster ; let c^* whose mean $\bar{c}$ is least distant from f, where distance is again measured based on weighted Jaccard distance: $J(\bar{c}, f)$. Then, if file f resides on a machine m that is expected to have singletons (that is, $g(m) = H$ as defined in Section 3.4.1), we filter it as benign iff $J(c^*, f) \leq \theta_H$; otherwise, it is filtered iff $J(c^*, f) \leq \theta_N$, where θ_H and θ_N are the corresponding filtering thresholds.

We select different θ values for the training and final versions of our pipeline. For training, our primary goal is to reduce the 80:1 benign to malicious class imbalance ratio so that we can train an effective classifier, whereas for testing, our goal is to achieve a high

[1] Note that this clustering of files is entirely distinct from the clustering of files in machine profiling, where non-singleton files are clustered based on machines that they appear on.

true positive rate while minimizing false positives. For purposes of creating a balanced training set, we select $\theta_N = 0.1$ and $\theta_H = 0.3$, which filters 91.8% of benign singletons, resulting in a more manageable 9:1 class imbalance ratio, at the cost of 7% of malware samples being thrown out of our training set. However this does not affect the performance of our model adversely, since during testing we can be less aggressive with the thresholds and pass more files to the classifier. In practice, we found values around $\theta_N = 0.07$ and $\theta_H = 0.13$ result in the best performance over the test data.

2.4.3 Malicious singleton detection

Having filtered out a large portion of predicted benign file instances, we are left with a residual data set of benign and malicious files that we classify using supervised-learning techniques. Though the filtering of benign files by the previous stages of our pipeline provide better class balance, we found that significant improvements in classification accuracy result when the residual data set is augmented by including 3 benign files that we sample randomly from each cluster C generated in the file profiling step. Doing so improves the classifier by adding additional benign files that are representative of the overall population of benign singleton files. We trained multiple classification algorithms with different strengths to determine which would be most effective at singleton classification.

Feature engineering is also key to the performance of our classifiers. Whereas machine and file profiling were designed for a backend system where a global view of the distribution of benign and malicious singleton files is available, here we design a classifier that we can deploy on client machines, based entirely on the static features of the file. Hence, we assume direct access to the files themselves and can build rich feature sets over the files, so long as they are not expensive to compute. This is in contrast to the telemetry used for machine and file profiling, for which network bandwidth constraints and privacy concerns limited the telemetry that could be collected. As mentioned in Section 3.3.3, we make no attempt to reverse the effects of obfuscation attempts employed by malware, finding that

the use of the obfuscation techniques themselves provides strong discriminative power that helps us to disambiguate between benign and malicious singletons.

Features

The features used by our learning algorithms to classify singleton program binary files fall into four categories.

1. The first category of features corresponds to features of file name and path. For these we used the same file name and path bag-of-words feature representation here as in the file profiling step of Section 3.4.2. To reduce the number of features included in our model, we applied a chi-squared feature selection to choose the most discriminative features [108].
2. The second category of features are derived from the header information of the executable file. We include all fields in the headers that are common to most windows executable files that exhibit some variability (some header fields never change). These header fields include the MS-DOS Stub, Signature, the COFF File Header, and the Optional Header (which is optional but nearly always present) [109].
3. We derive features from the Section Table found in the file's header, which describes each section of the binary, and also compute the entropy of each of the file's sections as features.
4. Our third category of features is derived from the external libraries that are dynamically linked to the program binary file. To determine which libraries the file links to, we create a feature for each of the most popular Windows library files (primarily Windows API libraries) that represents the number of functions imported from the library. We also create binary features for the individual functions in common Windows libraries that are most commonly used by malware. These take a value of 1 when the function is imported and 0 otherwise.

In all, category 1's bag of words features for filename and path consist of 300 features, while category 2,3, and 4 features together comprise close to 1000 features.

Classification

We apply two learning models, a Recurrent Neural Network (RNN) [110] and a Support Vector Machine with a radial basis function as its kernel [111], and compare their performance and ability to withstand adversarial manipulation in Section 3.5. The RNN model is particularly suited for textual data, so we train it solely using file names and path information as features. Given the sequential properties of the file name text, RNNs aim to make use of the dependency relationship among characters to classify malicious vsḃenign singletons. The goal of the character-level learning model is to predict the next character in a sequence and thereby classify the entire sequence based on the character distribution. Here, given a training sequence of characters $(a_1, a_2, ..., a_m)$, the RNN model uses the sequence of its output vectors $(o_1, o_2, ..., o_m)$ to obtain a sequence of distributions $P(a_{k+1}|a_{\leq k}) = softmax(o_k)$, where the softmax distribution is defined by

$$P(softmax(o_k) = j) = exp({o_k}^{(j)}) / \sum_k exp({o_k}^{(l)}). \tag{2.1}$$

The learning model's objective is to maximize the total log likelihood of the training sequence, which implies that the RNN learns a probability distribution over the character sequences used in a full path + filename. Then based on the benign and malicious character distribution patterns, the RNN model learns to classify the malicious singletons from the benign based on our balanced training data.

For the SVM model, we apply the text chunking technique described in Section 3.4.2, and use the bag-of-words representation as described above, concatenated with static and API-based features, where relevant. While numerous other classification algorithms could be used here, our purpose of exploring RNN and SVM specifically is to contrast an approach specifically designed for text data (making use of filename and path information exclusively) with a general-purpose learning algorithm that is known to perform well in malware classification settings [112].

Putting Everything Together

The high-level algorithm for the entire training pipeline is shown in Algorithm 1.

Algorithm 1 Train($\{S_{tr}, Z_{tr}, M_{tr}, Y_{tr}\}$):

1: $g = \text{machineProfiling}(\{S_{tr}, Z_{tr}, M_{tr}, Y_{tr}\})$
2: $(D, \theta_H, \theta_N, C) = \text{fileProfiling}(\{S_{tr}, Z_{tr}, M_{tr}, Y_{tr}\}, g)$
3: $h = \text{learnClassifier}(D)$
4: **return** $g, h, \theta_H, \theta_N, C$

The input to this algorithm is a collection of tuples $\{s_i, z_i, m_i, y_i\} \in \{S, Z, M, Y\}$ describing file instances on machines, which are partitioned into training (tr) and testing (te) for the pipeline. Each file instance is represented by s_i, the 256-bit digest of a SHA-2 hash over its contents and the size z_i of the file in bytes. The machine is represented by a unique machine identifier m_i, and each instance of the file receives a label y_i, which designates a file as benign, malicious, or unknown. Machine profiling processes the file-instance data to identify singleton files (those for which only one instance exists) from more prevalent software that it groups into packages and uses to predict the presence or absence of singletons. The end result of training the pipeline includes the two classifiers: g classifies machines into H (has benign singletons) and N (no benign singletons), while h classifies files as malicious or benign, trained based on the selected representative data D. Additional by-products include, the clusters of benign files C and the thresholds θ_H and θ_N that determine how aggressively files projected to be benign are filtered before the classifier h is applied.

Our test-time inputs include a set of singleton files that we withheld from training and our model parameters, and it returns simply whether or not to label f as benign or malicious. The specifics of the associated testing process, which use of our training pipeline, are given in Algorithm 2.

Algorithm 2 Predict($\{S_{te}, M_{te}\}$, $g, h, \theta_H, \theta_N, C$):

$l = g(M_{te})$: label the machine as H or N
$c^* = \arg\min_{c \in C} J(S_{te}, c)$ { find closest cluster center to S_{te} }
if $J(S_{te}, c) \leq \theta_l$ **then**
    return B {"benign" if S_{te} is close to a benign cluster center }
end if
return $h(S_{te})$ { otherwise, apply the classifier}

2.5 Experimental Evaluations

We conduct experiments on a large real-world dataset, dataset $D2$ as described in Section 3.3.1, to evaluate the proposed pipeline as well as analyze the robustness of learning system. As mentioned above, in implementing and deploying such a system in practice, we face a series of tradeoffs. The first is how much information about each file we should be collecting. On the one hand, more information will likely improve learning performance. On the other hand, collecting and analyzing data at such scale can become extremely expensive, both financially and computationally. Moreover, collection of detailed data about files on end-user machines can become a substantial privacy issue. For all of these reasons, very little information is traditionally collected about files on end-user systems, largely consisting of file name and an anonymized path, as well as file hashes and machines they reside on. For a subset of files, deeper information is available, including static features as well as API calls, as discussed above. However, these involve a significant cost: for example, extracting API calls requires static analysis. Our experiments are therefore designed to assess how much value these additional features have in classification, and whether or not it is truly worthwhile to be collecting them at the scale necessary for practical deployment. Our evaluation applies Machine Profiling (MP), File Profiling (FP), an RNN based on only file name features, a SVM based on file name features, a SVM based on both file name and the static features (SVMS), and a SVM based on file name, static features, and API function features (SVMSF).

2.5.1 Baseline Evaluation

Our first efficacy study demonstrates the benefit provided by our machine learning pipeline as compared to two natural baselines. Our first baseline applies machine and file profiling, ranking all examples based on their similarity to benign files, and identifying the samples that are furthest from benign cluster centers as malicious. Our second baseline is our best-performing classifier trained over our entire feature set (SVMSF), but trained without the benefit of an initial machine/file profiling step, which reduces the ratio of benign to malicious files from an 80:1 ratio to a 9:1 ratio. This baseline is similar to prior work in malware classification based on static features [95]. As seen in Figure 3.6, our full pipeline demonstrates clear improvement over the two baselines, with a significantly higher AUC score. The spot on the curve with the maximal $F_{0.5}$ score achieves a 92.1% true positive rate at a 1.4% false positive rate, a dramatic improvement over applying FP or SVMSF on its own. Different locations on the ROC curve are achieved by selecting increasing values for θ_N and θ_H. The maximal $F_{0.5}$ score is achieved with $\theta_H = 0.13$ and $\theta_N = 0.07$.

Though uninformed downsampling of benign files may reasonably be suggested as an alternative means to reduce the class imbalance and achieve better classification results with SVMSF, our attempts to do so resulted in classifiers that perform worse than the SVMSF classifier of Figure 3.6. The reason for this is likely that downsampling decimates small clusters of benign files, resulting in a model that represents benign singletons only by its most massively populated clusters. Our pipeline can be thought of as providing an informed downsampling of benign files that reduces massively populated clusters of benign files to a few prototypes, allowing the SVM to train a model that represents the full gamut of benign singletons with the additional benefit of doing so over a more balanced dataset.

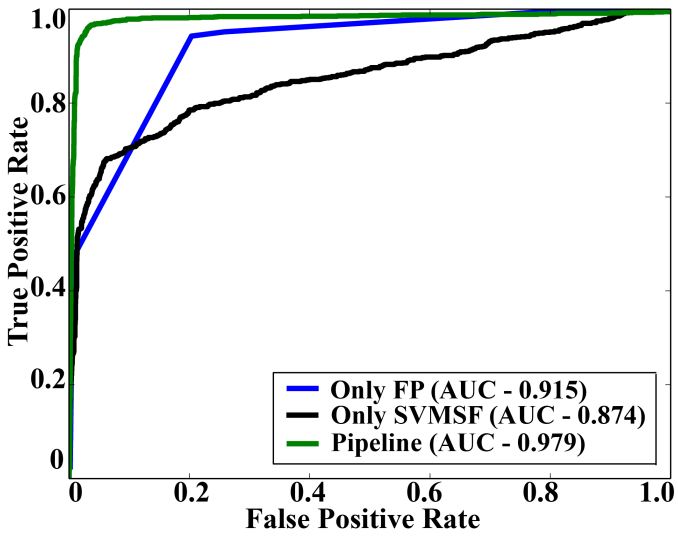


Figure 2.6: ROC-curve comparison of the pipeline performance with the two baselines: no machine/file profiling, and only machine/file profiling.

2.5.2 Evaluating Performance of the Classification Step

To assess the relative importance of the three classes of features (text, static, and API) used by our model, we analyze the relative performance of just the last classification step of four models on the dataset produced by MP and FP filtering: 1) RNN (using text features only), 2) SVM (using text features only), 3) SVM with both text and static features, and 4) SVM with text, static, and API features.

To highlight the performance differences between these classifiers, we evaluate them over a test set of singletons from which obviously benign singletons have been pre-filtered by file profiling (for this reason this figure does not reflect the overall performance of our pipeline as reported in Figure 3.6). Our first observation is that RNN outperforms SVM when only textual features are used, which is not surprising, given that RNN's are particularly well suited to text data. Second, our model's performance drops when training over filename and anonymized path plus static features, which demonstrates the high discriminative value of the filename and anonymized path relative to features derived from header information in the executable. However, these static features do offer value when we ac-

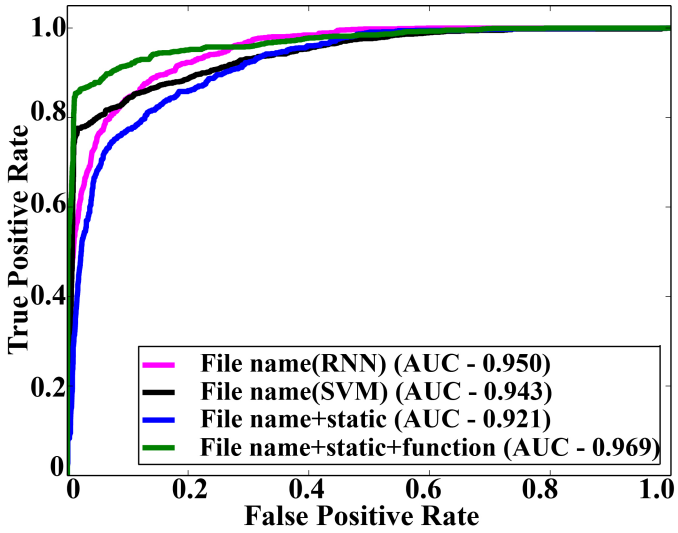


Figure 2.7: Comparisons for models with different features without attacker.

count for the potential for adversarial manipulation, as discussed in Section 3.5.3. Third, the value of features based on imported API functions is evident in the performance of the SVMSF model compared to all other models, particularly when we choose a threshold that limits the false positive rate, as security vendors are prone to do: The precision and recall scores that produce a maximal $F_{0.5}$ score for SVMSF are 83% recall at a 1% false positive rate, as compared to 76% recall at a 5% false positive rate for RNN, which is this model's closest competitor on an Area Under the Curve (AUC) basis. Note that the performance of the full pipeline is better than either of these classifiers alone (see Figure 3.6), because many of the benign files that are causing the FPs are labeled correctly using the machine and file profiling steps. Finally, our adversarial evaluation of these classifiers (Section 3.5.3) offers additional justification for incorporating static and imported function-based features into our model.

We evaluated the run-time required to train each step of our pipeline, including Machine Profiling (MP), File Profiling (FP), and the selected classifier, which is one of the following: RNN, SVM (based on only file name), SVMS, and SVMSF. The run-time of each step, when performed on a single powerful machine, is illustrated in Figure 3.8. Training Machine Profiling and File Profiling is fairly expensive, However, these two steps can

be done offline, and updated incrementally as new data arrives. Training the SVM classifiers is inexpensive, whereas training the RNN takes on the order of three hours with GPU acceleration. Though we do not believe that this is a cause for concern, the inferior performance of the RNN as compared to SVMSF makes it less appealing for inclusion in the final version of our pipeline. We do not include test-time performance evaluation since the cost to test a single file is negligible for all stages of the pipeline.

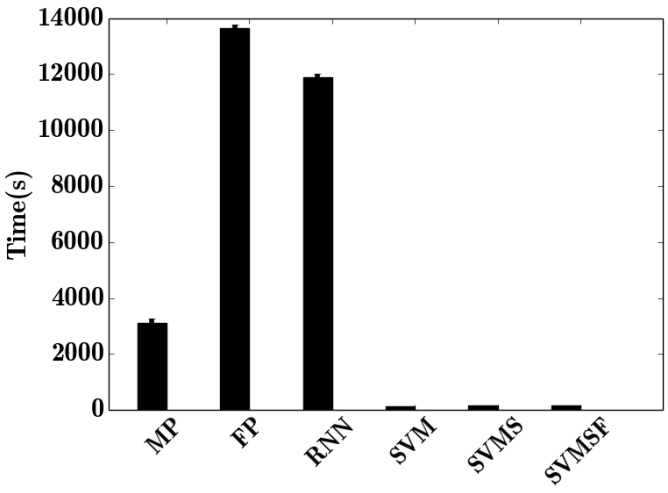


Figure 2.8: Comparisons of the runtime of different components within the pipeline.

2.5.3 Adversarial Evaluation

Though the evaluation of our classifiers, presented in Figure 3.7 is fairly typical for a malware classification tool, it is not necessarily indicative of the long-term performance of a classifier once it has been massively deployed in the wild. In particular, what is missing is an evaluation of the ability of our classifier to withstand the inevitable attempts of malware authors to respond to its deployment by modifying their malicious singleton files to mimic benign file patterns in order to evade detection. Whereas researchers have traditionally discussed an algorithm's robustness to evasion based on subjective arguments about the strength or weakness of individual features, the now well-developed body of research on *adversarial machine learning* provides more rigorous methods for evaluating the

adversarial robustness of a machine learning method [44, 45], and provides guidelines for developing more adversarially robust learning techniques [113, 114].

We proceed by providing an evaluation of our model's adversarial robustness. The adversarial resistance of a classifier evaluation presupposes a given classifier, h, that outputs for a given feature vector x, a label $h(x) \in \{-1, +1\}$, where in our case, -1 represents a benign prediction and $+1$ represents a malicious prediction. Given h, the adversary is modeled as aiming to minimize the cost of evasion,

$$x^* = \arg \min_{x' | h(x') = -1} c(x, x'),$$

where $c(x, x')$ is the cost of using a malicious instance x' in place of x to evade h (by ensuring that $h(x') = -1$, that is, that the malicious file will be classified as benign). The optimal evasion is represented by x^*. Because this model always results in a successful evasion, no matter its cost, we follow a more realistic model presented by Li and Vorobeychik [115], where the evasion only occurs when its cost is within a fixed adversarial budget B, thus: $c(x, x^*) \leq B$. Similarly, we mainly focus on the binary features here and prioritize the ones that have the most distinguished values for malicious and benign to modify, focusing the adversaries budget on the features that will be most useful for them to modify under the assumption that they known how to mimic benign software. In effect, we assume that the adversary will evade detection only if the gains from doing so outweigh the costs. The budget represents the percentage of the total number of features that the attacker is able to modify. A natural measure of the evasion cost $c(x, x')$ is the weighted l_1 distance between x and x': $c(x, x') = \sum_i a_i \|x_i - x'_i\|$. The choices of weights can be difficult to determine in a principled way, although some features will clearly be easier for an adversary to modify than others. We use $a_i = 1$ for all features i below as a starting point. As we will see, this already provides us with substantial evidence that a classifier using solely filename-based features is extremely exploitable by an adversary, without even accounting for the fact that

such features are also easier to modify for malware authors than, say, the functions they import from the Windows API and other libraries.

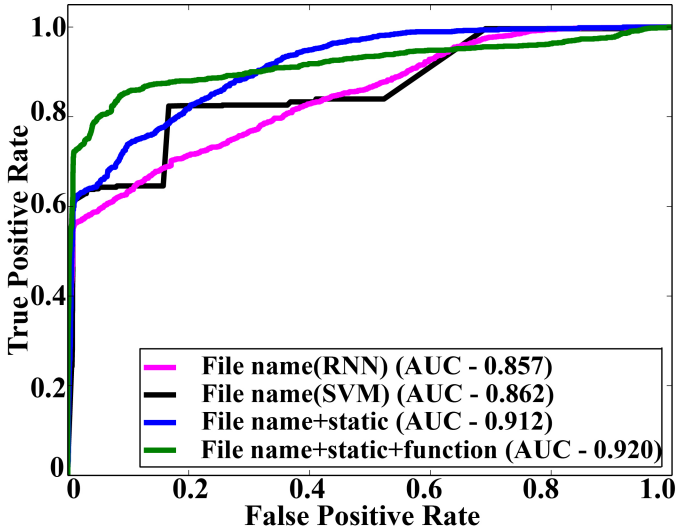


Figure 2.9: Comparisons for models with attacker budget as 5.

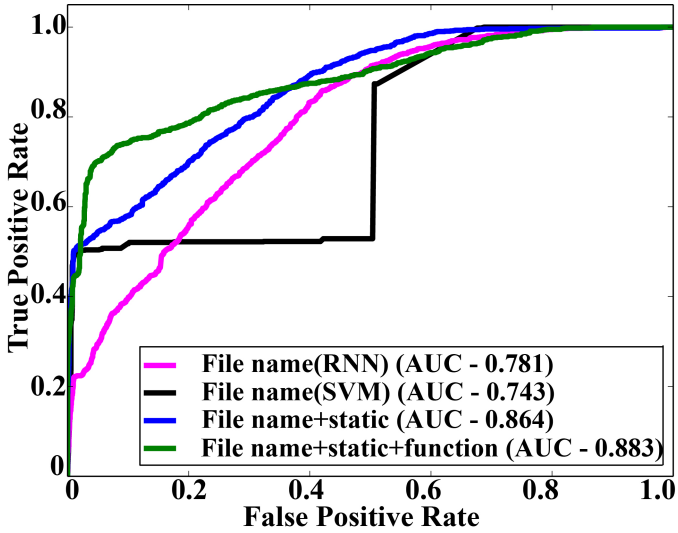


Figure 2.10: Comparisons for models with attacker budget as 10.

We now perform a comparison of the same classifier and feature combinations presented in Section 3.5.2, but we now evaluate these classifiers using evasion attacks, as shown in Figures 3.9 and 3.10 with budgets $B = 5\%$ and $B = 10\%$, respectively. These figures highlight a significant trend: whereas the RNN's performance was previously rather close to that of the SVM with filename, static, and imported function features, the former

has displays poor adversarial resistance, while the latter is far more robust. The RNN's AUC drops to 0.857 under pressure from a weaker attacker, and to 0.78 when pressured by a stronger one, whereas the AUC for the SVM with the largest feature set only drops to 0.92 under a smaller adversarial budget, and to 0.88 with a larger one). The SVM based only on filename features performed even worse than the RNN. Interestingly, while adding static features (and not imported function features) to the SVM degrades its adversary-free performance, the classifier performs considerably better than the RNN and SVM with filename features, in the presence of an adversary.

In summary, our experimental results point consistently to the use of a Support Vector Machine with features derived from the filename, path, static properties of the file, and imported functions, as the model that performs the best, even against an active adversary. Thus, the best version of our overall pipeline leverages this support vector machine as its classifier, achieving the overall performance results shown in Figure 3.6.

2.6 Summary of Contributions

We analyzed a large dataset to extract insights about the properties and distribution of singleton program binary files and their relationships to non-singleton software. We leverage the *context* in which singletons appear to filter benign files from our dataset, allowing us to train a model over a more balanced set of positive and negative examples. We build a classifier and feature set over the *static contents* of the file to effectively label benign and malicious singletons, in a way that is adversarial robust. Together, these components of our pipeline classify singletons much more effectively than either a context or a content-based approach can do on its own.

In summary, this study has the following contributions:

1. provide the first detailed discussion of the role that benign polymorphism plays in making singleton file classification a challenging problem.
2. identify root causes of benign polymorphism and leverage these to develop a method

for filtering the most "obvious" benign files prior to applying malware classification methods.

3. develop an algorithm that classifies 92% of malicious singletons as such, at a 1.4% false positive rate. We do so purely on the basis of static file properties, despite extensive obfuscation in most malware files, which we make no attempt to reverse.
4. explore the adversarial robustness of multiple classification models to an important class of automated evasion/mimicry attacks, demonstrating the robustness of a performant set of features derived from static file properties.

Chapter 3

EVASION ATTACKS IN ADVERSARIAL ENVIRONMENTS

Motivated by the real-world adversarial security problems such as spam and malware detection, in this chapter we will look into some of the fundamental questions for *secure learning*, including the attack model, defensive strategies, as well as the interactions between the defender (learner) and adversaries. The core challenge in this class of applications is that adversaries are not static data generators, but make a deliberate effort to evade the classifiers deployed to detect them. We investigate both the problem of modeling the objectives of such adversaries, as well as the algorithmic problem of accounting for rational, objective-driven adversaries. In particular, we demonstrate severe shortcomings of feature reduction in adversarial settings using several natural adversarial objective functions, an observation that is particularly pronounced when the adversary is able to substitute across similar features (for example, replace words with synonyms or replace letters in words). By exploring the properties and effects of evasion attacks in adversarial environments, we motivate the following work to develop robust learning algorithms against such evasion behaviors.

3.1 Overview

The success of machine learning has led to its widespread use as a workhorse in a wide variety of domains, from text and language recognition to trading agent design. It has also made significant inroads into security applications, such as fraud detection, computer intrusion detection, web search, and comparison shopping [9, 10, 11, 12]. The use of machine (classification) learning in security settings has especially piqued the interest of the research community in recent years because traditional learning algorithms are highly

susceptible to a number of attacks [25, 116, 117, 118, 119]. The class of attacks that is of interest to us are *evasion* attacks, in which an intelligent adversary attempts to adjust their behavior so as to evade a classifier that is expressly designed to detect it [25, 45].

Machine learning has been an especially important tool for filtering spam and phishing email, which we treat henceforth as our canonical motivating domain. To date, there has been extensive research investigating spam and phish detection strategies using machine learning, most without considering adversarial modification [120, 121, 122]. Failing to consider an adversary, however, exposes spam and phishing detection systems to evasion attacks. Typically, the predicament of adversarial evasion is dealt with by repeatedly re-learning the classifier. This is a weak solution, however, since evasion tends to be rather quick, and re-learning is a costly task, since it requires one to label a large number of instances (in crowdsourced labeling, one also exposes the system to deliberate corruption of the *training* data). Therefore, several efforts have focused on proactive approaches of modeling the learner and adversary as players in a game in which the learner chooses a classifier or a learning algorithm, and the attacker modifies either the training or test data [44, 123, 124, 125, 45, 84, 126].

Spam and phish detection, like many classification domains, tends to suffer from the curse of dimensionality [121]. Feature reduction is therefore standard practice, either explicitly, by pruning features which lack sufficient discriminating power, implicitly, by using regularization, or both [127]. One of our key novel insights is that in adversarial tasks, feature selection can open the door for the adversary to evade the classification system. This metaphorical door is open particularly widely in cases where *feature cross-substitution* is viable. By feature cross-substitution, we mean that the adversary can accomplish essentially the same end by using one feature in place of another. Consider, for example, a typical spam detection system using a "bag-of-words" feature vector. Words which in training data are highly indicative of spam can easily be substituted for by an adversary using synonyms or through substituting characters within a word (such replacing an "o"

with a "0"). We support our insight through extensive experiments, exhibiting potential perils of traditional means for feature selection.

3.2 Preliminaries

The Learner Let $X \subseteq \mathbb{R}^n$ be the feature space, with n the number of features. For a feature vector $x \in X$, we let x_i denote the ith feature. Suppose that the training set (x, y) is comprised of feature vectors $x \in X$ generated according to some unknown distribution $x \sim \mathscr{D}$, with $y \in \{-1, +1\}$ the corresponding binary labels, where the meaning of -1 is that the instance x is benign, while $+1$ indicates a malicious instance. The learner's task is to learn a classifier $g : X \to \{-1, +1\}$ to label instances as malicious or benign, using a training data set of labeled instances $\{(x_1, y_1), \ldots, (x_m, y_m)\}$.

The Adversary We suppose that every instance $x \sim \mathscr{D}$ corresponds to a fixed label $y \in \{-1, +1\}$, where a label of $+1$ indicates that this instance x was generated by an adversary. In the context of a threat model, therefore, we take this malicious x to be an expression of *revealed preferences* of the adversary: that is, x is an "ideal" instance that the adversary would generate if it were not marked as malicious (e.g., filtered) by the classifier. The core question is then what *alternative* instance, $x' \in X$, will be generated by the adversary. Clearly, x' would need to evade the classifier g, i.e., $g(x') = -1$. However, this cannot be a sufficient condition: after all, the adversary is trying to accomplish some goal. This is where the ideal instance, which we denote x^A comes in: we suppose that the ideal instance achieves the goal and consequently the adversary strives to limit deviations from it according to a cost function $c(x', x^A)$. Therefore, the adversary aims to solve the following optimization problem:

$$\min_{x' \in X : g(x') = -1} c(x', x^A). \tag{3.1}$$

There is, however, an additional caveat: the adversary typically only has query access to $g(x)$, and queries are costly (they correspond to actual batches of emails being sent out,

for example). Thus, we assume that the adversary has a fixed query budget, B_q. Additionally, we assume that the adversary also has a cost budget, B_c so that if the solution to the optimization problem (4.1) found after making B_q queries falls above the cost budget, the adversary will use the ideal instance x^A as x', since deviations fail to satisfy the adversary's main goals.

The Game The game between the learner and the adversary proceeds as follows:

1. The learner uses training data to choose a classifier $g(x)$.

2. Each adversary corresponding to malicious feature vectors x uses a query-based algorithm to (approximately) solve the optimization problem (4.1) subject to the query and cost budget constraints.

3. The defender's "test" error is measured using a new data set in which every malicious $x \in X$ is replaced with a corresponding x' computed by the adversary in step 2.

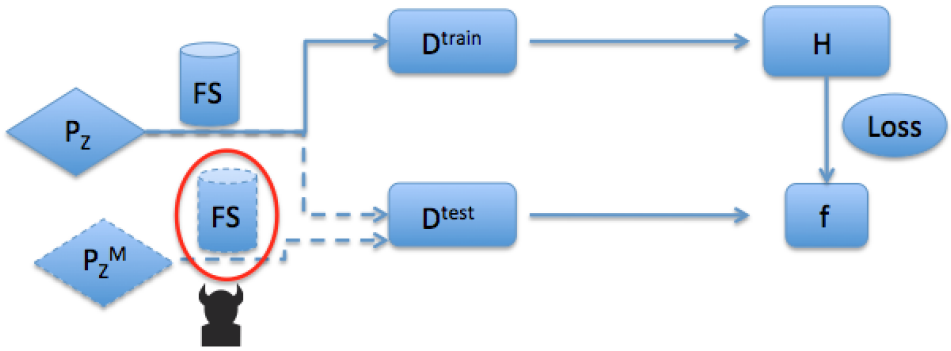


Figure 3.1: Genearl idea of feature cross-substitution attacks

3.3 Modeling Feature Cross-Substitution

Distance-Based Cost Functions In one of the first attempts at modeling adversaries in classification settings, Lowd and Meek [45] proposed a natural l_1 distance-based cost

function which penalizes for deviations from the ideal feature vector x^A:

$$c(x', x^A) = \sum_i a_i |x'_i - x^A_i|, \tag{3.2}$$

where a_i is a relative importance of feature i to the adversary. All follow-up work in the adversarial classification domain has used either this cost function, or variations of distance, such as l_p norms [25, 116, 119, 128].

Feature Cross-Substitution Attacks While distance-based cost functions seem eminently natural models of adversarial objective, they miss an important phenomenon of feature cross-substitution. In spam or phishing, this phenomenon is most obvious when an adversary substitutes words for their synonyms or substitutes similar-looking letters in words. As an example, consider Figure 4.2 (left), where some features can naturally be

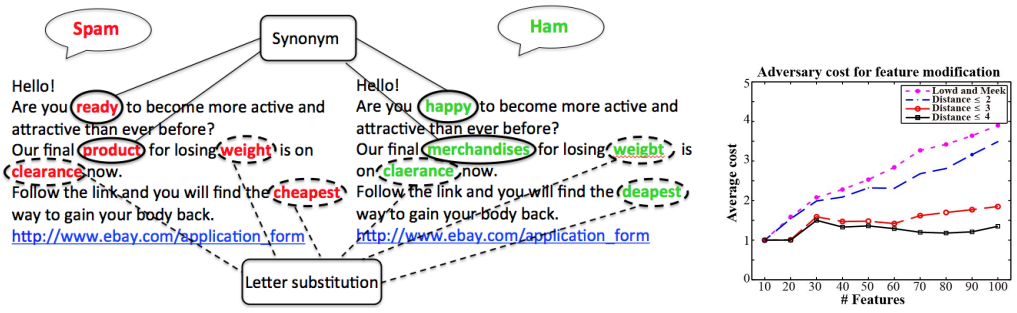


Figure 3.2: Left: illustration of feature substitution attacks. Right: comparison between distance-based and equivalence-based cost functions.

substituted for others without significantly changing the original content. These words can contain features with the similar meaning or effect (e.g. *money* and *cash*) or differ in only a few letters (e.g *clearance* and *claerance*). The impact is that the adversary can achieve a much lower cost of transforming an ideal instance x^A using similarity-based feature substitutions than simple distance would admit.

To model feature cross-substitution attacks, we introduce for each feature i an equivalence class of features, F_i, which includes all admissible substitutions (e.g., k-letter word modifications, synonyms, etc), and generalize the Lowd and Meek cost function to account

for such cross-feature equivalence:

$$c(x', x^A) = \sum_i \min_{j \in F_i | x_j^A \oplus x'_j = 1} a_i |x'_j - x_i^A|, \tag{3.3}$$

where $x_j^A \oplus x'_j = 1$ ensures that we do not "double-count" the same substitution. Figure 4.2 (right) shows the cost comparison between the Lowd and Meek and equivalence-based cost functions under letter substitution attacks. The key observation is that the equivalence-based cost function significantly reduces attack costs compared to the distance-based cost function, with the difference increasing in the size of the equivalence class. The practical import of this observation is that the adversary will far more frequently come under cost budget when he is able to use such substitution attacks. Failure to capture this phenomenon therefore results in a threat model that significantly underestimates the adversary's ability to evade a classifier.

3.4 The Perils of Feature Reduction in Adversarial Classification

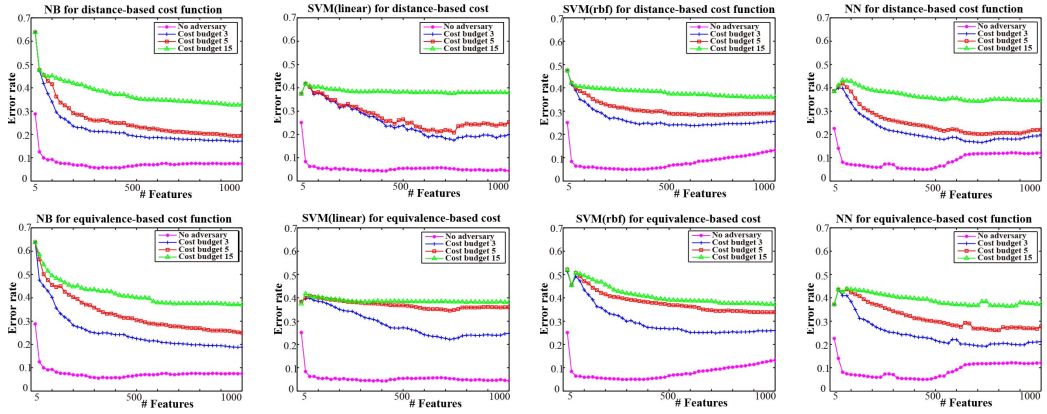
Feature reduction is one of the fundamental tasks in machine learning aimed at controlling overfitting. The insight behind feature reduction in traditional machine learning is that there are two sources of classification error: bias, or the inherent limitation in expressiveness of the hypothesis class, and variance, or inability of a classifier to make accurate generalizations because of overfitting the training data. We now observe that in adversarial classification, there is a crucial third source of generalization error, introduced by *adversarial evasion*. Our main contribution in this section is to document the tradeoff between feature reduction and the ability of the adversary to evade the classifier and thereby introduce this third kind of generalization error. In addition, we show the important role that feature cross-substitution can play in this phenomenon.

To quantify the perils of feature reduction in adversarial classification, we first train each classifier using a different number of features n. In order to draw a uniform com-

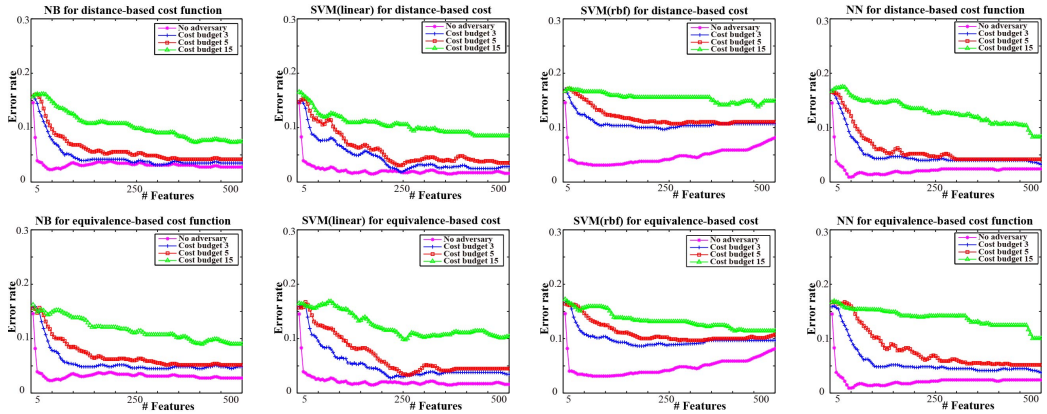
parison across learning algorithms and cost functions, we used an algorithm-independent means to select a subset of features given a fixed feature budget n. Specifically, we select the set of features in each case based on a score function $score(i) = |FR_{-1}(i) - FR_{+1}(i)|$, where $FR_C(i)$ represents the frequency that a feature i appears in instances x in class $C \in \{-1, +1\}$. We then sort all the features i according to score and select a subset of n highest ranked features. Finally, we simulate an adversary as running an algorithm which is a generalization of the one proposed by Lowd and Meek [45] to support our proposed equivalence-based cost function (see the supplement Section 2 for details).

In our evaluation we consider three data sets: Enron email data [129], Ling-spam data [130], and internet advertisement dataset from the UCI repository [131]. The Enron data set was divided into training set of 3172 and a test set of 2000 emails in each of 5 folds of cross-validation, with an equal number of spam and non-spam instances [129]. A total of 3000 features were chosen for the complete feature pool, and we sub-selected between 5 and 1000 of these features for our experiments. The Ling-spam data set was divided into 1158 instances for training and 289 for test in cross-validation and contains 1000 features from which between 5 and 500 were sub-selected for the experiments. Finally, the UCI data set was divided into 476 training and 119 test instances in five-fold cross validation, with four times as many advertisement as non-advertisement instances. This data set contains 200 features, of which between 5 and 200 were chosen. For each data set, we compared the effect of adversarial evasion on the performance of four classification algorithms: Naive Bayes, SVM with linear and rbf kernels, and neural network classifiers.

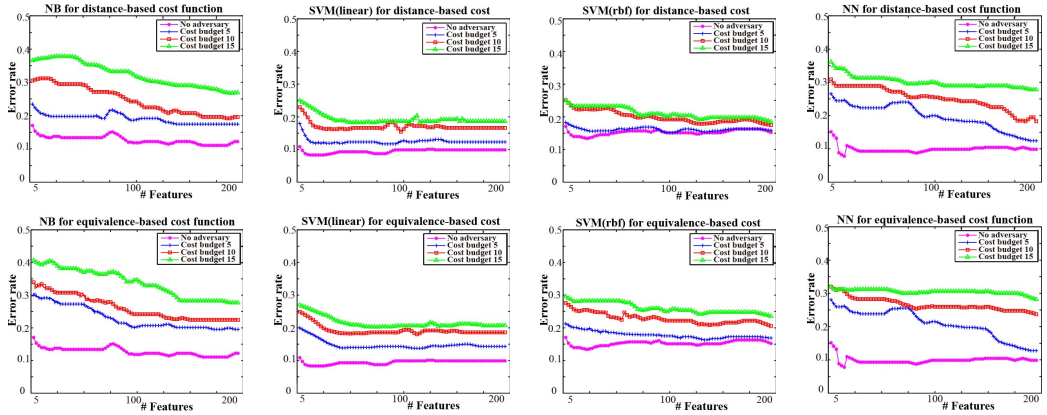
The results are documented in Figure 4.3. To understand the results, we can consider the lowest (purple) lines in all plots, which show cross-validation error as a function of the number of features used, as the baseline comparison. In most cases, there is an "optimal" number of features, i.e., the point at which the cross-validation error rate reaches a minimum, and we can presume that traditional machine learning methods will strive to select the number of used features near this point. The first key observation is that whether the



1(a) 1(b) 1(c) 1(d)



2(a) 2(b) 2(c) 2(d)



3(a) 3(b) 3(c) 3(d)

Figure 3.3: Effect of adversarial evasion on feature reduction strategies. (a)-(d) deterministic Naive Bayes classifier, SVM with linear kernel, SVM with rbf kernel, and Neural network, respectively. 1-3 correspond to Enron, Ling-spam, and UCI data sets. Top sets of figures in each case correspond to distance-based and bottom figures are equivalence-based cost functions. For equivalence-based cost functions equivalence classes are formed using max-2-letter substitutions.

adversary uses the distance- or equivalence-based cost functions, there tends to be a shift of this "optimal" point to the right: the learner should be using *more* features when facing a threat of adversarial evasion, despite the potential risk of overfitting. The second observation is that when a significant amount of malicious traffic is present (as in the experiments based on Enron data), evasion can account for a dominant portion of the test error, shifting the error up significantly. Third, feature cross-substitution attacks often exhibit distinctly more power, significantly elevating test error, particularly when the cost budget is sufficiently high. This impact is particularly stark as we increase the size of the equivalence class (as documented in the supplement Section 1).

3.5 Summary of Contributions

In this chapter we offer two solutions against the evasion attacks in adversarial classification. The first is highly heuristic, using meta-features constructed using feature equivalence classes for classification. The second is a principled and general Stackelberg game multi-adversary model (SMA), solved using mixed-integer linear programming. We use experiments to demonstrate that the first solution often outperforms state-of-the-art adversarial classification methods, while SMA is significantly better than all alternatives in all evaluated cases. We also show that SMA in fact implicitly makes a tradeoff between feature reduction and adversarial evasion, with more features used in the context of stronger adversaries.

In summary, the contributions are:

1. A comprehensive defensive strategy analysis for evasion attacks,
2. a heuristic class-based learning approach (Section 5.2), and
3. a bi-level optimization framework and solution methods that make a principled tradeoff between feature selection and adversarial evasion (Section 5.3).

Chapter 4

STACKELBERG GAME MULTI-ADVERSARY DEFENSIVE MODEL

To defend the evasion attacks introduced before, in this chapter we offer a simple heuristic method for making learning more robust to feature cross-substitution attacks. We then present a more general approach based on mixed-integer linear programming with constraint generation, which implicitly trades off overfitting and feature selection in an adversarial setting using a sparse regularizer along with an evasion model. This approach is the first method for combining an adversarial classification algorithm with a very general class of models of adversarial classifier evasion. We will also show that our algorithmic approach significantly outperforms state-of-the-art alternatives.

4.1 Overview

Aiming to defend the evasion attacks, especially the feature-substitute attacking strategy, we propose several defensive strategies in this chapter to handle the classification problem in the adversarial environments. The first proposed solution to the problem of feature reduction in adversarial classification is *equivalence-based learning*, or constructing features based on feature equivalence classes, rather than the underlying feature space. We show that this heuristic approach does, indeed, significantly improve resilience of classifiers to adversarial evasion. Our second proposed solution is more principled, and takes the form of a general bi-level mixed integer linear program to solve a Stackelberg game model of interactions between a learner and a collection of adversaries whose objectives are inferred from training data. The baseline formulation is quite intractable, and we offer two techniques for making it tractable: first, we cluster adversarial objectives, and second, we use constraint generation to iteratively converge upon a locally optimal solution. The prin-

cipal merits of our proposed bi-level optimization approach over the state of the art are: a) it is able to capture a very general class of adversary models, including the model proposed by Lowd and Meek [45], as well as our own which enables feature cross-substitution; in contrast, state-of-the-art approaches are specifically tailored to their highly restrictive threat models; and b) it makes an implicit tradeoff between feature selection through the use of sparse (l_1) regularization and adversarial evasion (through the adversary model), thereby solving the problem of adversarial feature selection. The general model of this problem is illustrated in Figure 4.1.

4.2 Equivalence-Based Classification

Having documented the problems associated with feature reduction in adversarial classification, we now offer a simple heuristic solution: equivalence-based classification (EBC). The idea behind EBC is that instead of using underlying features for learning and classification, we use equivalence classes in their place.[1] Specifically, we partition features into equivalence classes. Then, for each equivalence class, we create a corresponding meta-feature to be used in learning. For example, if the underlying features are binary and indicating a presence of a particular word in an email, the equivalence-class meta-feature would be an indicator that *some* member of the class is present in the email. As another example, when features represent frequencies of word occurrences, meta-features could represent aggregate frequencies of features in the corresponding equivalence class.

4.3 Stackelberg Game Multi-Adversary Model

The proposed equivalence-based classification method is a highly heuristic solution to the issue of adversarial feature reduction. We now offer a more principled and general approach to adversarial classification based on the game model described above. Formally,

[1] We assume throughout that equivalence classes are pre-defined. The question of learning these from data is left for future work.

we aim to compute a Stackelberg equilibrium of the game in which the learner moves first by choosing a linear classifier $g(x) = w^T x$ and all the attackers simultaneously and independently respond to g by choosing x' according to a query-based algorithm optimizing the cost function $c(x', x^A)$ subject to query and cost budget constraints. Consequently, we term this approach *Stackelberg game multi-adversary model (SMA)*. The optimization problem for the learner then takes the following form:

$$\min_w \alpha \sum_{j|y_j=-1} l(-w^T x_j) + (1-\alpha) \sum_{j|y_j=1} l(w^T F(x_j; w)) + \lambda ||w||_1, \tag{4.1}$$

where $l(\cdot)$ is the hinge loss function and $\alpha \in [0,1]$ trades off between the importance of false positives and false negatives. Note the addition of l_1 regularizer to make an explicit trade-off between overfitting and resilience to adversarial evasion. Here, $F(x_j; w)$ generically captures the adversarial decision model. In our setting, the adversary uses a query-based algorithm (which is an extension of the algorithm proposed by Lowd and Meek [45]) to approximately minimize cost $c(x', x_j)$ over $x' : w^T x' \leq 0$, subject to budget constraints on cost and the number of queries. In order to solve the optimization problem (5.1) we now describe how to formulate it as a (very large) mixed-integer linear program (MILP), and then propose several heuristic methods for making it tractable. Since adversaries here correspond to feature vectors x_j which are malicious (and which we interpret as the "ideal" instances x^A of these adversaries), we henceforth refer to a given adversary by the index j.

The first step is to observe that the hinge loss function and $\|w\|_1$ can both be easily linearized using standard methods. We therefore focus on the more challenging task of expressing the adversarial decision in response to a classification choice w as a collection of linear constraints.

To begin, let $\bar{X}$ be the set of all feature vectors that an adversary can compute using a fixed query budget (this is just a conceptual tool; we will not need to know this set in practice, as shown below). The adversary's optimization problem can then be described as

computing

$$z_j = \underset{x' \in \bar{X} | w^T x' \leq 0}{\arg\min} \; c(x', x_j)$$

when the minimum is below the cost budget, and setting $z_j = x_j$ otherwise. Now define an auxiliary matrix T in which each column corresponds to a particular attack feature vector x', which we index using variables a; thus T_{ia} corresponds to the value of feature i in attack feature vector with index a. Define another auxiliary binary matrix L where $L_{aj} = 1$ iff the strategy a satisfies the budget constraint for the attacker j. Next, define a matrix c where c_{aj} is the cost of the strategy a to adversary j (computed using an arbitrary cost function; we can use either the distance- or equivalence-based cost functions, for example). Finally, let z_{aj} be a binary variable that selects exactly one feature vector a for the adversary j. First, we must have a constraint that $z_{aj} = 1$ for exactly one strategy a: $\sum_a z_{aj} = 1 \; \forall \; j$. Now, suppose that the strategy a that is selected is the best available option for the attacker j; it may be below the cost budget, in which case this is the strategy used by the adversary, or above budget, in which case x_j is used. We can calculate the resulting value of $w^T F(x_j; w)$ using $e_j = \sum_a z_{aj} w^T (L_{aj} T_a + (1 - L_{aj}) x_j)$. This expression introduces bilinear terms $z_{aj} w^T$, but since z_{aj} are binary these terms can be linearized using McCormick inequalities [132]. To ensure that z_{ja} selects the strategy which minimizes cost among all feasible options, we introduce constraints $\sum_a z_{aj} c_{aj} \leq c_{a'j} + M(1 - r_{a'})$, where M is a large constant and $r_{a'}$ is an indicator variable which is 1 iff $w^T T_{a'} \leq 0$ (that is, if a' is classified as benign); the corresponding term ensures that the constraint is non-trivial only for a' which are classified benign. Finally, we calculate r_a for all a using constraints $(1 - 2r_a) w^T T_a \leq 0$. While this constraint again introduces bilinear terms, these can be linearized as well since r_a are binary. The full MILP formulation is shown as below.

As is, the resulting MILP is intractable for two reasons: first, the best response must be computed (using a set of constraints above) for each adversary j, of which there could be many, and second, we need a set of constraints for each feasible attack action (feature vector) $x \in \bar{X}$ (which we index by a). We tackle the first problem by clustering the "ideal"

attack vectors x_j into a set of 100 clusters and using the mean of each cluster as x^A for the representative attacker. This dramatically reduces the number of adversaries and, therefore, the size of the problem. To tackle the second problem we use constraint generation to iteratively add strategies a into the above program by executing the Lowd and Meek algorithm in each iteration in response to the classifier w computed in previous iteration. In combination, these techniques allow us to scale the proposed optimization method to realistic problem instances. The MILP to compute solution to (5.1) is shown as below.

$$\min_{w,z,r} \alpha \sum_{i|y_i=0} D_i + (1-\alpha) \sum_{i|y_i=1} S_i + \lambda \sum_j K_j$$

$$\text{s.t.}: \quad \forall a,i,j : z_i(a), r(a) \in \{0,1\}$$

$$\sum_a z_i(a) = 1$$

$$\forall i : e_i = \sum_a m_i(a)(L_{ai}T_a + (1-L_{ai})x_i)$$

$$\forall a,i,j : -Mz_i(a) \leq m_{ij}(a) \leq Mz_i(a)$$

$$\forall a,i,j : w_j - M(1-z_i(a)) \leq m_{ij}(a) \leq w_j + M(1-z_i(a))$$

$$\forall a : \sum_j w_j T_{aj} \leq 2 \sum_j T_{aj} y_{aj}$$

$$\forall a,j : -Mr_a \leq y_{aj} \leq Mr_a$$

$$\forall a,j : w_j - M(1-r_a) \leq y_{aj} \leq w_j + M(1-r_a)$$

$$\forall i : D_i = \max(0, 1 - w^T x_i)$$

$$\forall i : S_i = \max(0, 1 + e_i)$$

$$\forall j : K_j = \max(w_j, -w_j)$$

The full SMA iterative algorithm using clustering and constraint generation is shown in Algorithm 3. The matrices L and C in the MILP can be pre-computed using the matrix of strategies and corresponding indices T in each iteration, as well as the cost budget B_c.

computeAttack() is the attacker's best response as in Algorithm 4.

Algorithm 3 SMA(X)

T =randStrats() // initial set of attacks

$X' \leftarrow$ cluster(X)

$w_0 \leftarrow$ MILP(X', T)

$w \leftarrow w_0$

while T changes **do**

  for $x^A \in X'_{spam}$ **do**

    t =computeAttack(x^A, w)

    $T \leftarrow T \cup t$

  end for

  $w \leftarrow$ MILP(X', T)

end while

return w

Here the algorithm 4 makes use of specific query budget and call the query algorithm to iteratively generate attack strategies based on both the query and cost budget.

Algorithm 4 computeAttack(x^A, w)

Get matrix T

Generate matrix C, L based on T, B_c

Randomly select x^- from X_{ham}

$B_q \leftarrow Q$

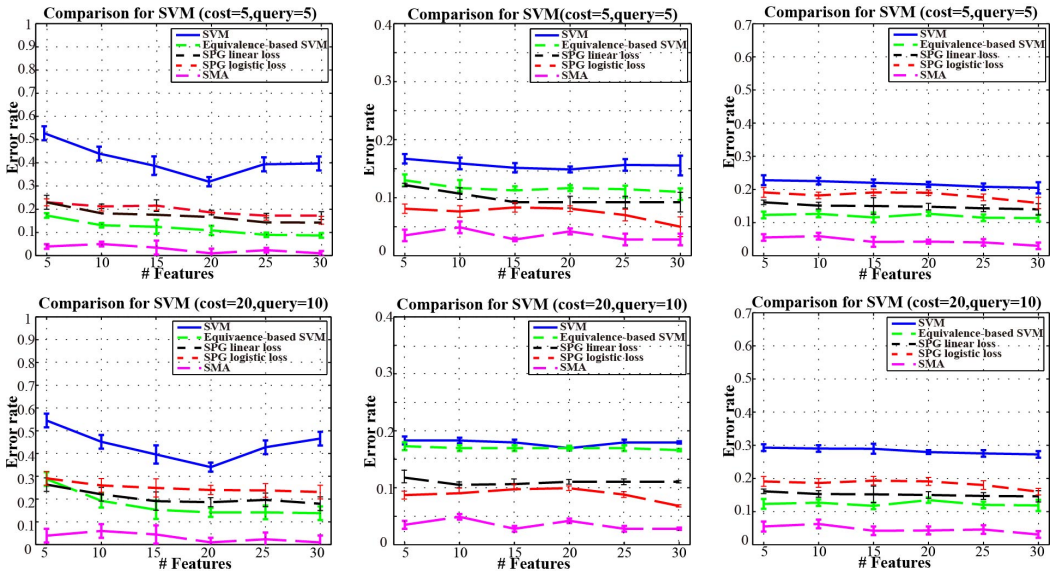
$t \leftarrow FindBooleanIMAC(x^A, x^-, w, B_q)$

return t

4.4 Experiments

In this section we investigate the effectiveness of the two proposed methods: the equivalence-based classification heuristic (EBC) and the Stackelberg game multi-adversary model (SMA) solved using mixed-integer linear programming. As in Section 4.4, we consider three data sets: the Enron data, Ling-spam data, and UCI data. We draw a comparison to three baselines: 1) "traditional" machine learning algorithms (we report the results for SVM; comparisons to Naive Bayes and Neural Network classifiers are provided in the section 5.4, 2) Stackelberg prediction game (SPG) algorithm with linear loss [84], and 3) SPG with logistic loss [84]. Both (2) and (3) are state-of-the-art alternative methods developed specifically for adversarial classification problems.

General evaluation of SMA Our first set of results, shown in Figure 5.1, is a performance comparison of our proposed methods to the three baselines, evaluated with respect to an adversary striving to evade the classifier, subject to query and cost budget constraints. In the case of the Enron data, we can see, remarkably, that the equivalence-based classi-



(a) (b) (c)

Figure 4.1: Comparison of EBC and SMA approaches to baseline alternatives on Enron data (a), Ling-spam data (b), and UCI data(c). Top: $B_c = 5, B_q = 5$. Bottom: $B_c = 20, B_q = 10$.

fier often significantly outperforms both SPG with linear and logistic loss. On the other hand, the performance of EBC is relatively poor on Ling-spam data, although observe that even the traditional SVM classifier has a reasonably low error rate in this case. While the performance of EBC is clearly data-dependent, SMA (purple lines in Figure 5.1) exhibits dramatic performance improvement compared to alternatives in all instances.

Figure 5.2 (left) looks deeper at the nature of SMA solution vectors w. Specifically, we consider how the adversary's strength, as measured by the query budget, affects the sparsity of solutions as measured by $\|w\|_0$. We can see a clear trend: as the adversary's budget increases, solutions become less sparse. This is to be expected in the context of our investigation of the impact that adversarial evasion has on feature reduction (Section 4.4): SMA automatically accounts for the tradeoff between resilience to adversarial evasion and regularization.

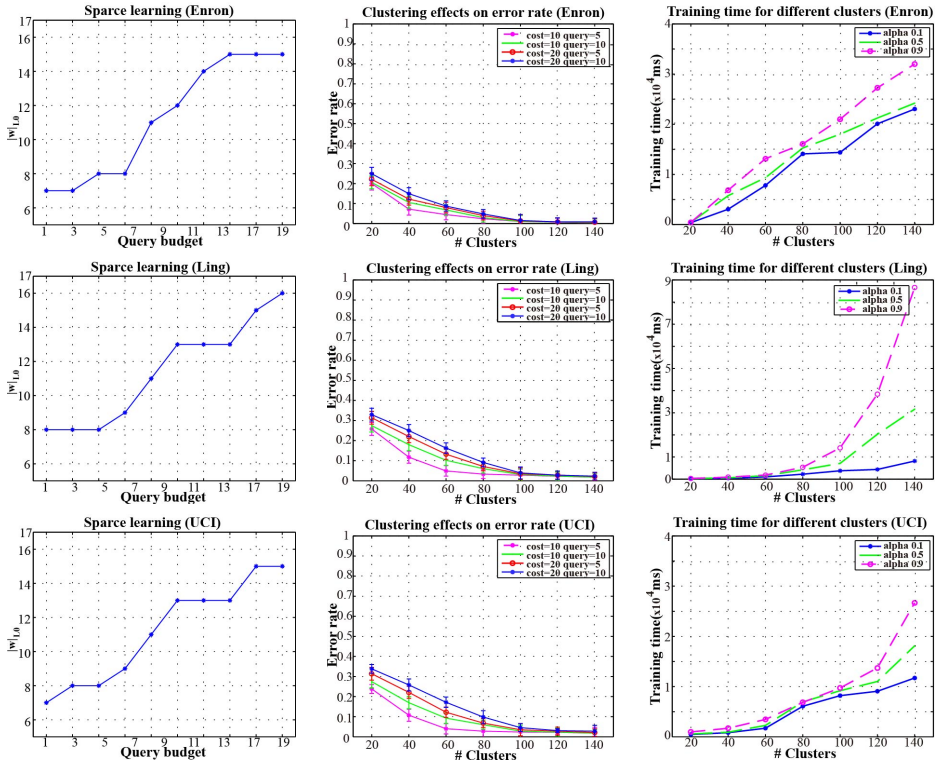


Figure 4.2: Left: $\|w\|_0$ of the SMA solution. Middle: SMA error rates, and Right: SMA running time, as a function of the number of clusters used. Top: results based on Enron data. Middle: results based on Ling data. Bottom: results based on UCI data.

Figure 5.2 (middle, right) considers the impact of the number of clusters used in solving the SMA problem on running time and error. The key observation is that with relatively few (80-100) clusters we can achieve near-optimal performance, with significant savings in running time.

Cost function influence Furthermore, we exhibit the comparison results to evaluate the effectiveness of the two proposed methods: the equivalence-based classification heuristic (EBC) and the Stackelberg game multi-adversary model (SMA) solved using mixed-integer linear programming. The evaluation is based on both the distance-based and equivalence-based cost functions.

We employ three datasets: the Enron data, Ling-spam data, and UCI data. Each column in Figure 5.3 to Figure 5.8 corresponds to a specific dataset. Figure 5.3 to Figure 5.5 show the comparison results for the Stackelberg prediction game (SPG) with linear loss, SPG with logistic loss, the two proposed methods, and each of the baseline classifier: Naive Bayes, SVM, and Neural Network respectively, based on equivalence-based cost function. Figure 5.6 to Figure 5.8 reveal the similar comparison results for the two SPG state-of-the-art alternatives and the proposed methods with each baseline classifier based on the distance-based cost function. Various cost (5, 10, 20) and query (5, 10) budget constraints are applied to simulate the adversarial evasion.

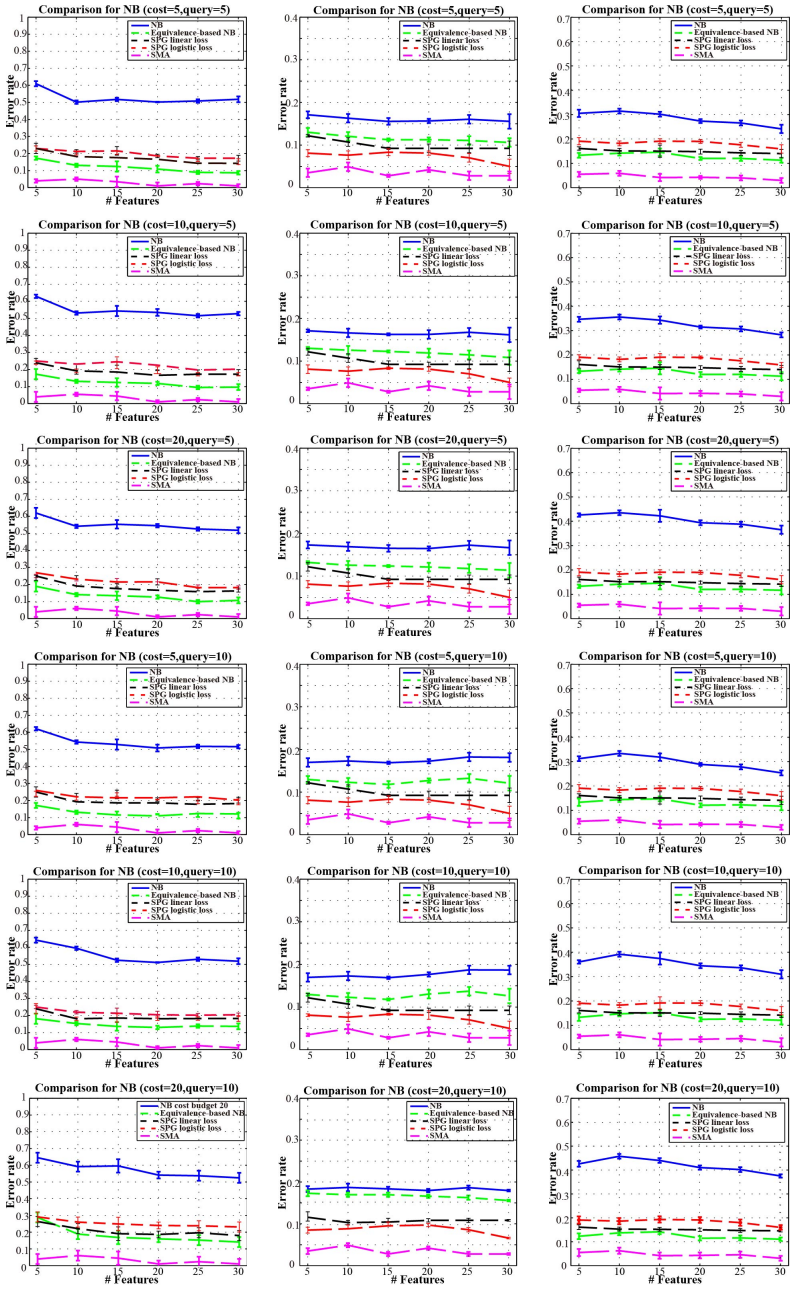
From Figure 5.3 to Figure 5.8, it is obvious that SMA outperforms other alternatives in all situations subject to various combinations of cost and query budget constraints based on different datasets. The performance of EBC is relatively data-dependent but still show resilience to the adversarial feature cross-substitution attacks compared with the traditional baseline classifiers. The comparison results also suggest that given higher cost and query budget, the adversary received stronger ability to perform feature cross-substitution attacks and therefore elevate the test error for the traditional classifiers, which fail to taken adversarial attacks into account. Furthermore, even having considered the adversarial settings for classification tasks, the test error rate of all classifiers based on the distance-based cost

function is still higher than the corresponding one based on the equivalence-based cost function. This implies that under estimate the adversary ability would lead to bad performance for classifiers. However, as SMA model can apply more robust cost function (equivalence-based cost function) to evaluate the adversary strategies accordingly during training, the test error of SMA is able to keep relatively stable for different attacked data, which highly increases the classifier robustness.

Comparison based on different equivalence class sizes

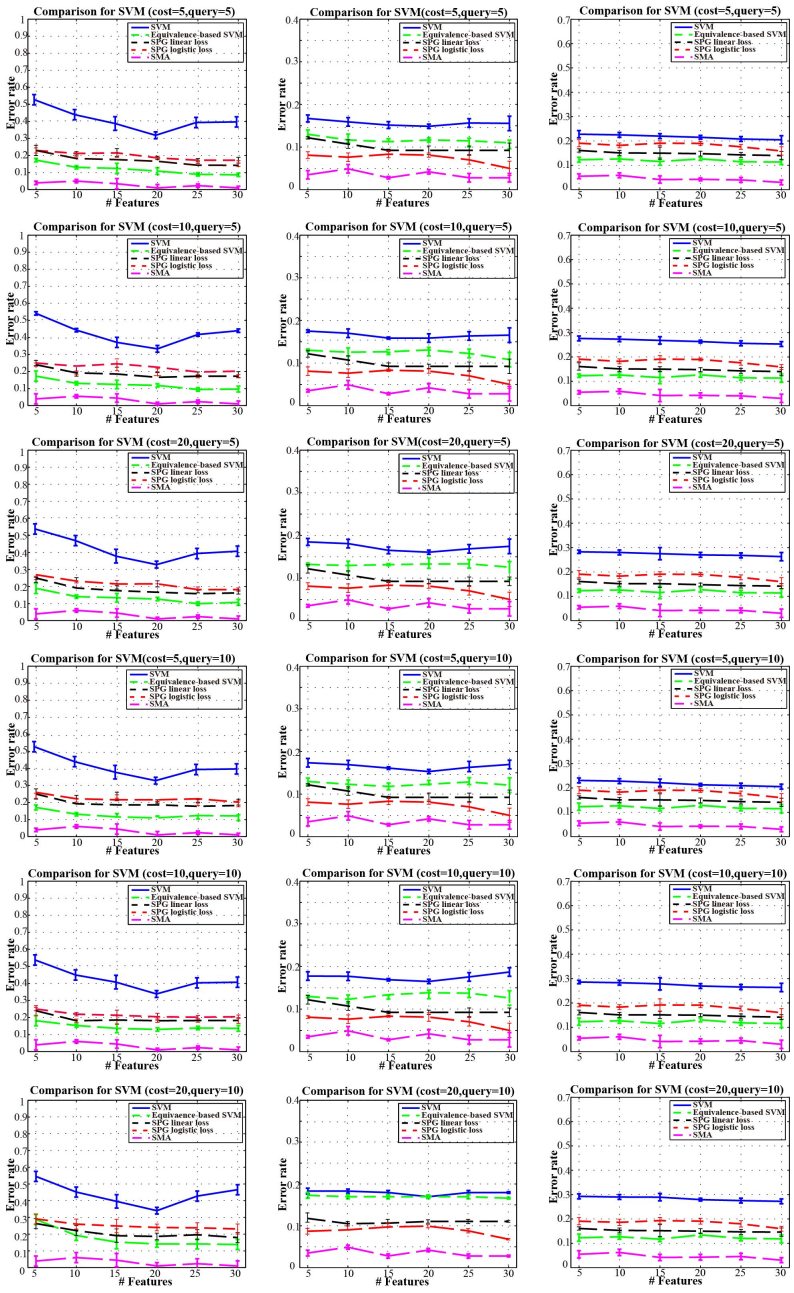
To demonstrate the impact of feature cross-substitution attacks, we show comparisons for NB, SVM with linear kernel, SVM with rbf kernel and Neural Network classifiers based on the baseline Distance-based 5.9 (a) and the Equivalence-based 5.9 (b)-(d) cost function with Enron data. For the equivalence-based cost function, we applied max-2,3,4-letter substitution respectively to form equivalence classes with increasing sizes. From the comparison results in Figure 5.9, it is obvious that the feature cross-substitution attacks elevate the test error on a large scale, and such attack gains more power when the equivalence class size increases.

Error injection evaluation To test the robustness of the proposed learning algorithm, we inject estimation errors for the defender to evaluate the learning results. We first allow the attacker to use the cost budget of 20 and query budgets as 3 and 30, respectively, while assume the defender incorrectly thought the adversarial cost is 5, 10, and 20 (no error). We also random flip the 10% of the elements within the L matrix of defender to simulate other unknown source of attacks to evaluate the robustness. When performing the attacks, we allow the adversary to attack the data in two ways: random and ranked. In the random attack, the adversary random chooses features to flip the sign until he meets the cost and query budgets. While in the ranked attack, the adversaries are more powerful and can have access to the feature score derived by the defender and attack the features from the highest score. In all kinds of attacks, the results stay robust without significant variants, which demonstrate the robustness of SMA in adversarial environments.



(a) (b) (c)

Figure 4.3: Comparison of EBC and SMA approaches to the baseline classifier Naive Bayes and SPG alternatives based on Equivalence-based cost function for (a) Enron data, (b)Ling-spam data, and (c) UCI data. Row 1: $B_c = 5, B_q = 5$, Row 2: $B_c = 10, B_q = 5$, Row 3: $B_c = 20, B_q = 5$, Row 4: $B_c = 5, B_q = 10$, Row 5: $B_c = 10, B_q = 10$, Row 6: $B_c = 20, B_q = 10$.



(a) (b) (c)

Figure 4.4: Comparison of EBC and SMA approaches to the baseline classifier SVM and SPG alternatives based on Equivalence-based cost function for (a) Enron data, (b)Ling-spam data, and (c) UCI data. Row 1: $B_c = 5, B_q = 5$, Row 2: $B_c = 10, B_q = 5$, Row 3: $B_c = 20, B_q = 5$, Row 4: $B_c = 5, B_q = 10$, Row 5: $B_c = 10, B_q = 10$, Row 6: $B_c = 20, B_q = 10$.

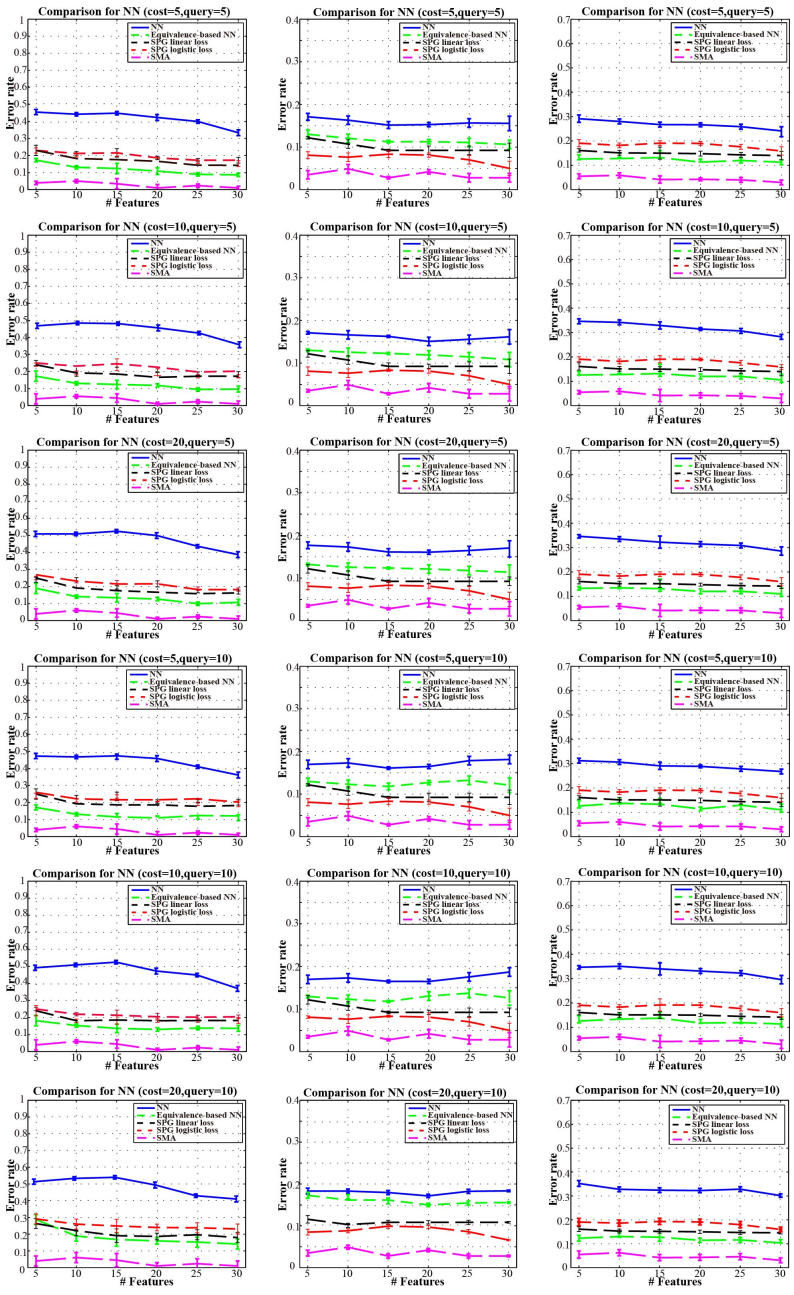


Figure 4.5: Comparison of EBC and SMA approaches to the baseline classifier Neural Network and SPG alternatives based on Equivalence-based cost function for (a) Enron data, (b)Ling-spam data, and (c) UCI data. Row 1: $B_c = 5, B_q = 5$, Row 2: $B_c = 10, B_q = 5$, Row 3: $B_c = 20, B_q = 5$, Row 4: $B_c = 5, B_q = 10$, Row 5: $B_c = 10, B_q = 10$, Row 6: $B_c = 20, B_q = 10$.

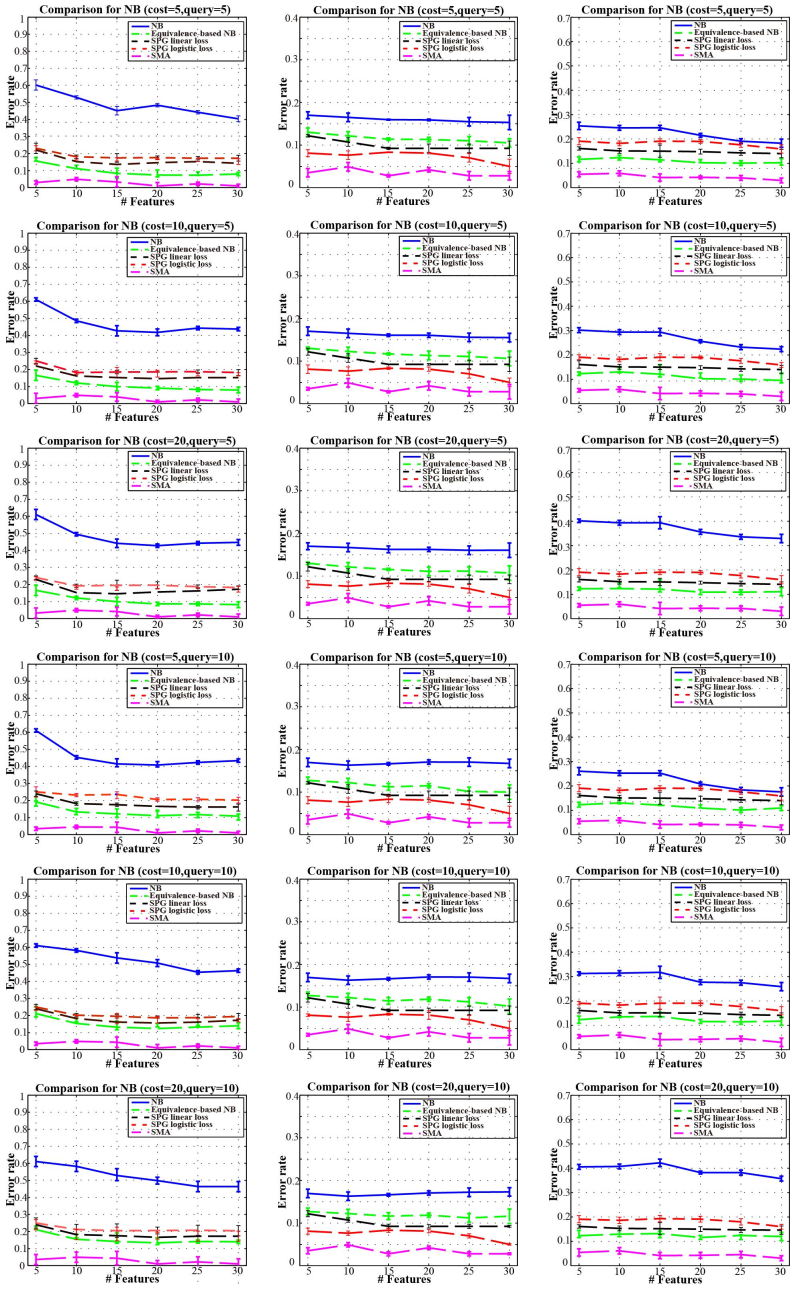


Figure 4.6: Comparison of EBC and SMA approaches to the baseline classifier Naive Bayes and SPG alternatives based on Distance-based cost function for (a) Enron data, (b)Ling-spam data, and (c) UCI data. Row 1: $B_c = 5, B_q = 5$, Row 2: $B_c = 10, B_q = 5$, Row 3: $B_c = 20, B_q = 5$, Row 4: $B_c = 5, B_q = 10$, Row 5: $B_c = 10, B_q = 10$, Row 6: $B_c = 20, B_q = 10$.

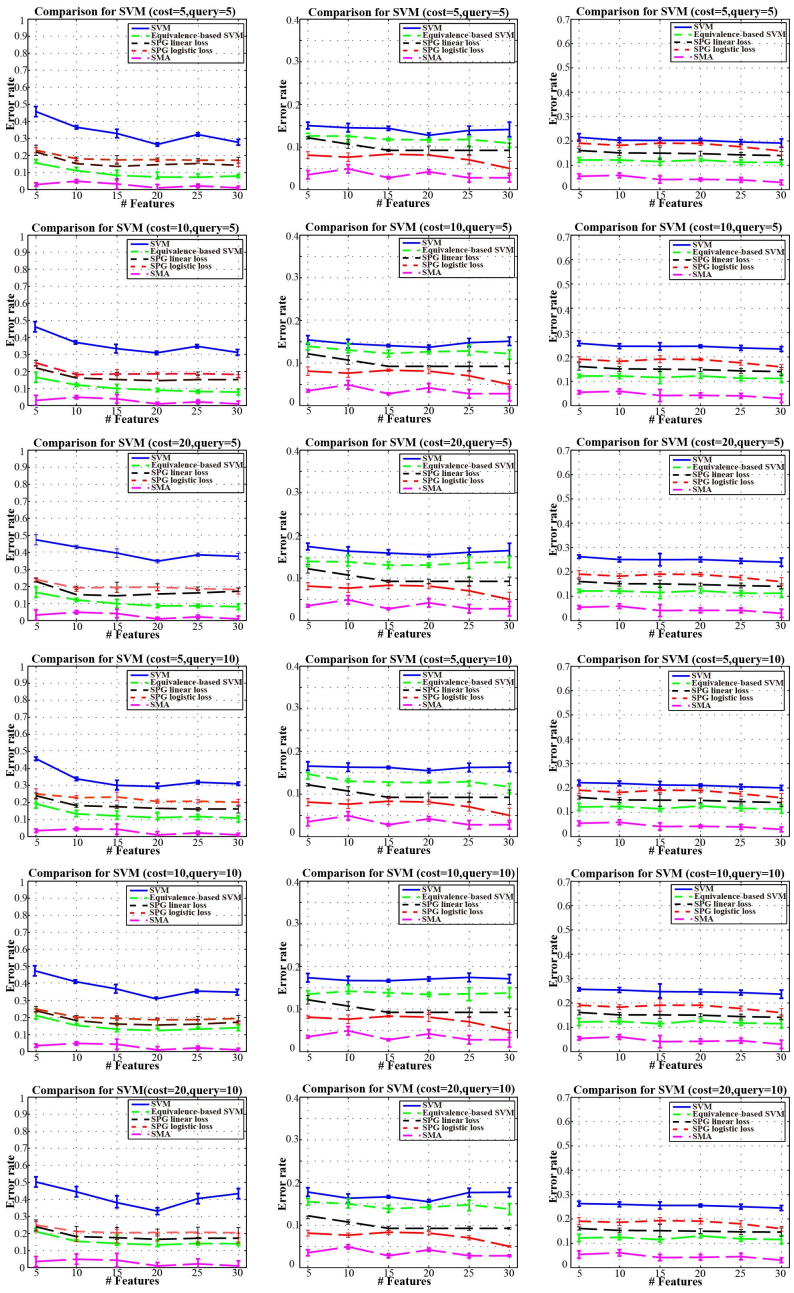


Figure 4.7: Comparison of EBC and SMA approaches to the baseline classifier SVM and SPG alternatives based on Distance-based cost function for (a) Enron data, (b)Ling-spam data, and (c) UCI data. Row 1: $B_c = 5, B_q = 5$, Row 2: $B_c = 10, B_q = 5$, Row 3: $B_c = 20, B_q = 5$, Row 4: $B_c = 5, B_q = 10$, Row 5: $B_c = 10, B_q = 10$, Row 6: $B_c = 20, B_q = 10$.

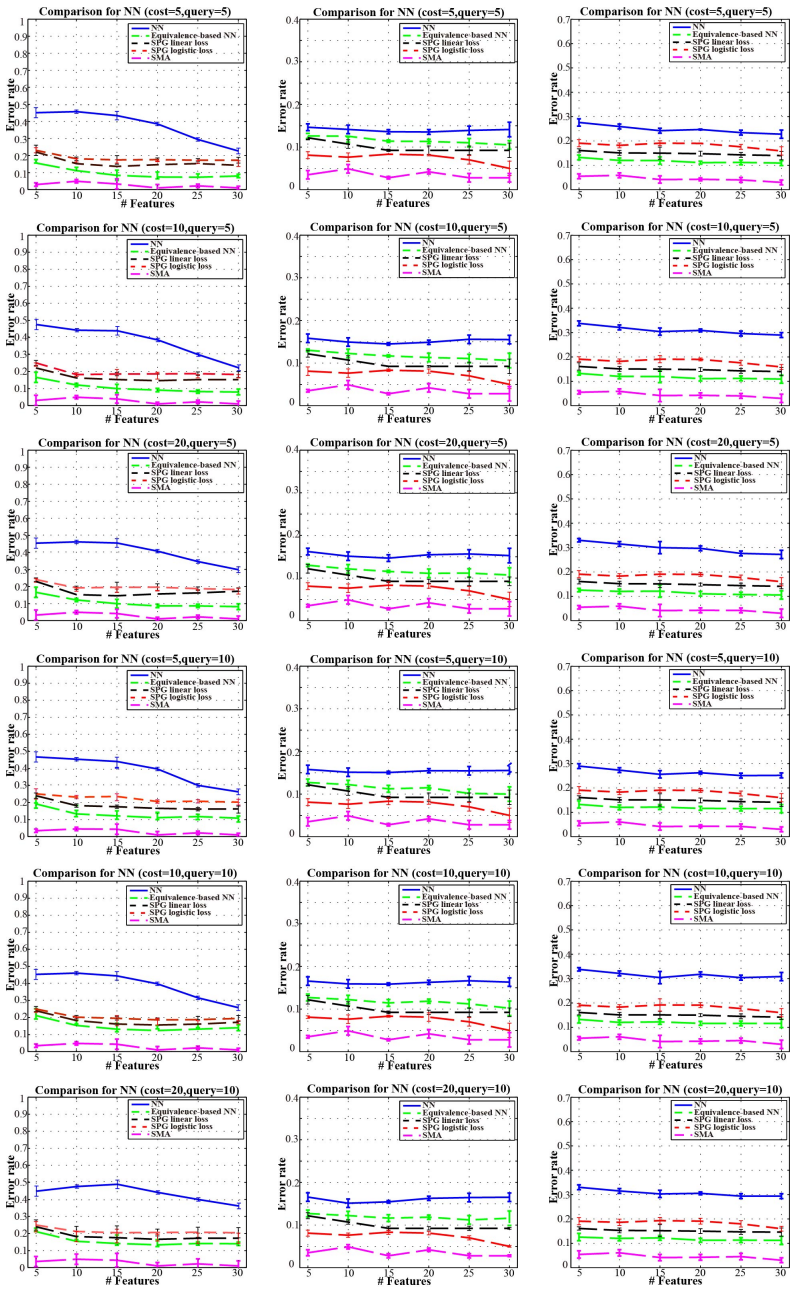
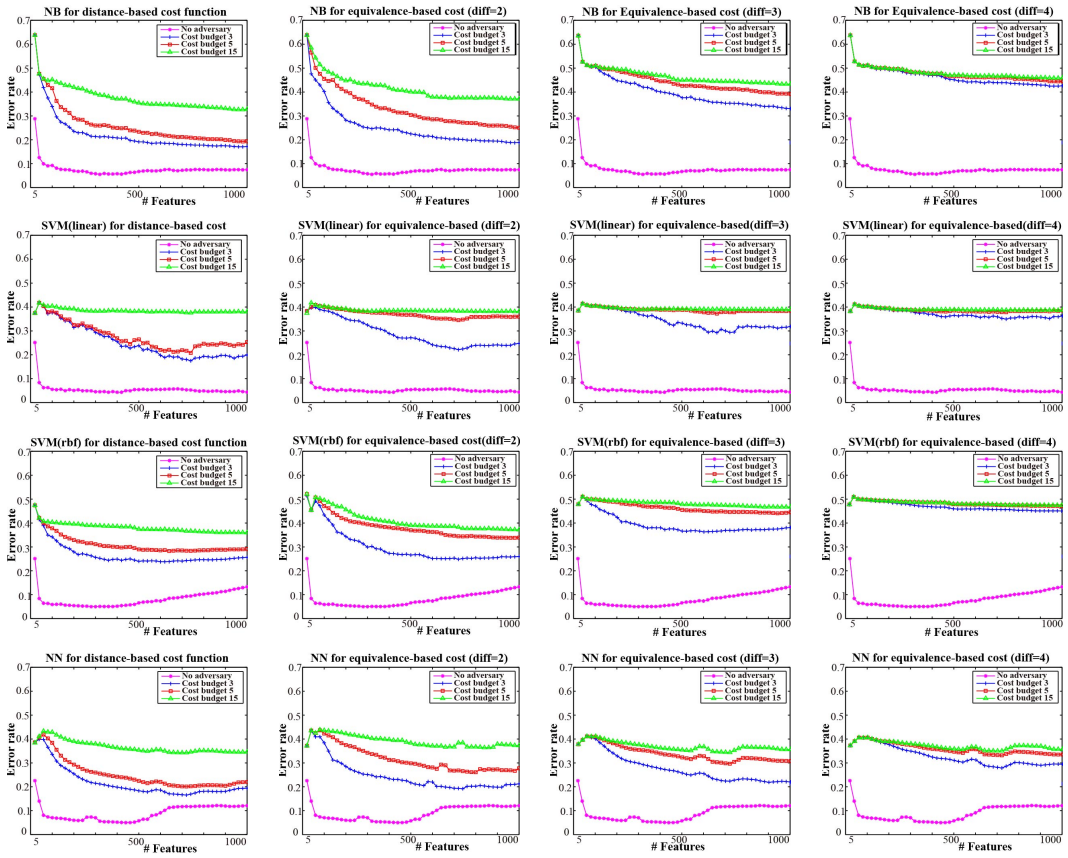


Figure 4.8: Comparison of EBC and SMA approaches to the baseline classifier Neural Network and SPG alternatives based on Distance-based cost function for (a) Enron data, (b)Ling-spam data, and (c) UCI data. Row 1: $B_c = 5, B_q = 5$, Row 2: $B_c = 10, B_q = 5$, Row 3: $B_c = 20, B_q = 5$, Row 4: $B_c = 5, B_q = 10$, Row 5: $B_c = 10, B_q = 10$, Row 6: $B_c = 20, B_q = 10$.



(a) (b) (c) (d)

Figure 4.9: Impacts of different equivalence class sizes for (a) Distance-based cost function, (b) Equivalence-based cost function with max-2-letter substitution, (c) Equivalence-based cost function with max-3-letter substitution, (d) Equivalence-based cost function with max-4-letter substitution.

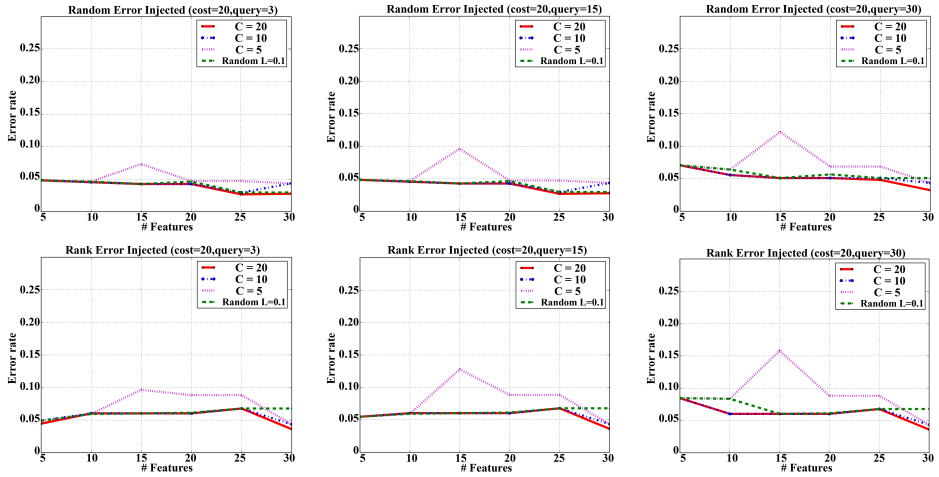


Figure 4.10: Estimation error injection for random feature attack (first line) and ranked features (second line).

4.5 Summary of Contributions

This chapter investigated two phenomena in the context of adversarial classification settings: classifier evasion and feature reduction, exhibiting strong tension between these. The tension is surprising: feature/dimensionality reduction is a hallmark of practical machine learning, and, indeed, is generally viewed as increasing classifier robustness. Our insight, however, is that feature selection will typically provide more room for the intelligent adversary to choose features not used in classification, but providing a near-equivalent alternative to their "ideal" attacks which would otherwise be detected. Terming this idea *feature cross-substitution* (i.e., the ability of the adversary to effectively use different features to achieve the same goal), we offer extensive experimental evidence that aggressive feature reduction does, indeed, weaken classification efficacy in adversarial settings.

In summary, the contributions are:

1. Explore the evasion attacks in adversarial environments,

2. a new adversarial evasion model that explicitly accounts for the ability to cross-substitute features (Section 4.3),

3. an experimental demonstration of the perils of traditional feature selection (Section 4.4).

Chapter 5

BEHAVIORAL EXPERIMENTS IN ADVERSARIAL SPAM FILTER EVASION

Previous research in adversarial machine learning all models the threat model mathematically without being able to derive the adversaries behavior patterns from the real dataset. In this chapter we start to conduct the human subject experiments aiming to simulate the attacker behavior from the real collected data and therefore model the threat model in a more practical way. The science of human decision making has seen much attention across a broad array of disciplines, but human decisions in complex adversarial settings such as cyber security are not well understood. In particular, here we study a particular decision problem of wide importance in cyber security: modification of an email message, such as a spam of phishing email, in order to evade a spam filter. Despite the obvious importance of understanding adversarial evasion of email filters, driven by the enormous social cost of spam and phishing attacks, ours is the first behavioral investigation into this problem. Within the analysis, we designed a human subject experiment in which subjects were presented with an "ideal" email instance, which they could modify in order to bypass a filter designed through supervised classification methods. One of our key findings is that adding a slight amount of noise to the filter significantly reduces the effectiveness of subjects to evade it. In addition to the analysis of experimental data, we use it to develop a model of human learning behavior in the context of our task. We found that our computational model is able to reliably replicate the experimental findings, suggesting that it can be used for future computational investigations into adversarial behavior, as well as in the design of email filtering algorithms that are robust to adversarial evasion.

5.1 Overview

Email has come to be a mainstay of our daily lives. According to a 2012 McKinsey report people spend on average 28% of workday activities using email. Nie *et al.* have suggested that an average email user loses 10 working days each year dealing with such emails [133], and most estimates have placed worldwide cost of spam to over 10 billion dollars [134].

Decades of research have yielded numerous methods for designing spam filters making use of knowledge engineering [135] and machine (typically, classification) learning [136, 137], with the latter having become the prominent paradigm [138]. In simple terms, the machine learning approach works by collecting many labeled instances of emails, where labels correspond to bad (spam) and normal (non-spam, sometimes called ham) emails. Emails themselves are represented quantitatively as feature vectors, with features often corresponding to presence or absence of specific indicator words or phrases, and a classification algorithm, such as Naive Bayes or Support Vector Machine [139] is run on the data to obtain a classifier which, given an arbitrary email instance (coded into features) outputs a decision whether this instance is spam or ham. For a given labeled data set, machine learning algorithms have come to be extremely good at detecting spam. The problem is that spammers have themselves become quite sophisticated in the techniques of *filter evasion*, or manipulating the spam email templates to bypass common filtering techniques. A typical approach in the field is a cat-and-mouse game in which a classifier is re-trained on new data at regular intervals, and spammers routinely change behavior in response [138]. Clearly, a more proactive approach is called for, and a literature emerged with a focus on modeling and algorithmic assessment of the *classifier (filter) evasion problem* as well as associated proactive learning algorithm design [44, 45, 114, 43, 113] In these highly stylized models, a spammer is typically viewed as aiming to minimize the number of edits to an "ideal" spam email template (for example, because modifications to the ideal instance adversely affect the associated response rate), subject to a hard constraint that the email

evades the filter. While much progress has been made in considering an explicit model of spammer evasion, the central limitation of all of this literature is that the models are not grounded in actual spammer behavior.

To address this limitation, we present the first human subject study of adversarial evasion of a classification-based spam filter. In our experiments, 265 human subjects, recruited using Amazon Mechanical Turk, each faced a task of editing an initial spam or phishing email template in order to achieve two objectives: first, bypass a filter, and second, remain close to the original. Our treatments were explicitly designed to investigate two hypotheses. The first hypothesis was that adding a small amount of noise to filtering decisions significantly reduces the subjects' ability to evade it. While prior work exists investigating the design of optimal randomization schemes in adversarial settings [140, 141], including work involving human subjects (e.g., [142]), ours is the first to investigate how randomization affects decision efficacy, and is also the first to consider randomization in experimental investigation of spam filter evasion. The second hypothesis was that measuring distance to the original in a way that does not penalize the subjects for making word substitutions, such as using synonyms, will significantly improve their performance. We find strong support for the first hypothesis, whereas, somewhat surprisingly, the support for the second hypothesis is mixed. In particular, we observe that randomized filtering significantly reduces the ability of subjects to evade, largely because it significantly increases the fraction of times that participants' submissions were filtered by the system. An additional finding of great practical importance is that increasing the fraction of words in an original email that are included as features in the classifier (filter) also increases the difficulty of the associated task. One of the core guiding principles in applied machine learning is that one should keep the number of features used as small as possible. Our finding, in contrast, suggests that when faced with an adversarial classification problem, such as spam filtering, limiting the number of features can actually make the evasion problem easier. In addition, we observed that female participants tend to perform better at the evasion task than males.

Besides our collection of experimental findings, we also endeavored to develop a synthetic model of evasion behavior calibrated using the experimental data. We demonstrate that our model effectively predicts both individual-level behavior, as well as aggregate experimental quantities, allowing us to successfully replicate the experimental findings in simulation. The developed model can thus be utilized in follow-up development of proactive spam filtering methods that explicitly account for human evasion behavior. The general structure of this human subject based experiments for cost evaluation is shown in Figure 6.1

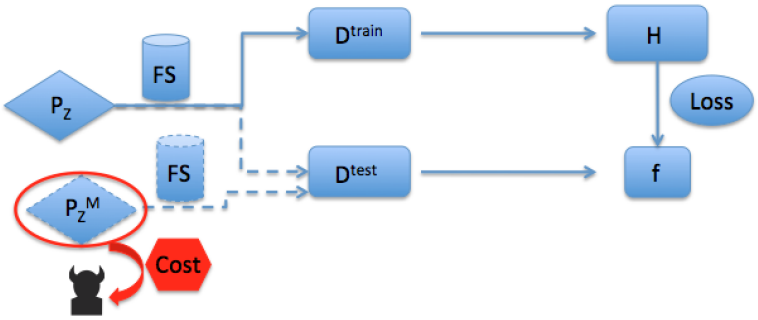


Figure 5.1: Genearl idea of human subject experiments on cost function evaluation

5.2 Preliminaries

Before we can properly describe and motivate the experimental setup, we begin by considering in the abstract the problem of *adversarial classifier evasion*, which has been previously studied in several forms from a computational standpoint [39, 45, 114, 43]. Spam and phishing email filtering is a special case: when machine learning is used to filter spam emails, the spammer changes the email structure (template) in order to evade the filter to ensure successful delivery of the email. The crucial challenge is that evasion alone is not sufficient: one needs to ensure that the original purpose of the email is still fulfilled. Next, we describe how this tradeoff can be quantified.

An instance of interest, such as an email (which may or may not be spam) is represented as a vector of binary features, $x = \{x_1, \ldots, x_n\}$. The corresponding label is encoded as either $+1$, signifying a malicious instance (spam, for example), or -1, signifying a normal or

benign instance (a regular email). In using machine learning for spam/phishing detection, we start with a training data set of instances $\{(x_i, y_i)\}$ where x_i are the feature vectors and y_i are the corresponding labels, and train a classifier, $g(x)$, which predicts a label for an arbitrary feature vector (email) x. Many techniques have been proposed for this problem. For our purposes, we used a Naive Bayes classifier, and used 500 features (words) that had the highest (R)elative (F)requency in spam / non-spam emails respectively. The relative frequency for each word i is defined as $|RF_{-1}(i) - RF_{+1}(i)|$, where $RF_C(i)$ is the relative frequency that word i appears in instances x in class C. We used TREC data to train the classifier with the average accuracy of $\sim$91% in five-fold cross-validation.

Given a classifier, $g(x)$, adversarial evasion is commonly modeled as (an adversary) choosing another feature vector x' (for example, a modified spam email template) which is classified as benign and is as close to the original instance x as possible [45, 128, 43]. Formally, let $c(x,x')$ be a cost function representing the loss sustained by the adversary from choosing x' instead of x. The adversary is solving the following optimization problem:

$$\min_{x'|g(x')=-1} c(x,x').$$

The simplest way to measure this cost is by using a norm, commonly, l_1. We define this cost function as:

$$S_1 : \; c(x,x') = \sum_i |x_i - x_i{}'|.$$

In recent work, this cost function has been criticized on the grounds that it penalizes for substitutions among equivalent features (for example, synonyms or 1-letter substitutions in words) [43]. This work proposed an alternative cost function defined as follows:

$$S_2 : \; c(x,x') = \sum_i \min_{j \in F_i | x_j{}' \oplus x_j = 1} |x_j - x_i{}'|,$$

where x_i denotes the i*th* feature within the instance x; F_i is the equivalence class (i.e., the set

of equivalent features) for a feature x_i; and $\oplus$ represents the exclusive or here to guarantee the features are substituted instead of only being deleted. In our experiments we defined the equivalence class of a word to be its synonyms (evaluated using the semantic dictionary *WordNet* [143]) and 1- and 2-character substitutions.

5.3 Experiment Design

Since we intend to study adversarial evasion of spam filters which use classification learning, our ideal source of subjects is spammers or phishers. It is clearly infeasible to obtain enough subjects from this population for an experiment. As a proxy, we use human subjects recruited using Amazon Mechanical Turk, a popular crowd-sourcing platform that is commonly utilized by behavioral science researchers to recruit and pay human subjects [144]. While not ideal, there is now substantial evidence that results from the experiments using Amazon Mechanical Turk are often indistinguishable from those found in physical laboratories [145, 146]. To collect data, we built a Rails application and ran it on the Amazon Web Services EC2, while storing data on Amazon Web Services RDS. In all, we recruited 265 participants for the study who have jointly completed 482 tasks (described below).

After signing up for the experiment, each participant received a simple and brief English language test (see the Supplement for details). Passing this test qualified them for participation in the experiment. At this point, subjects were invited to read the tutorial describing the experimental setup (see the Supplement for details). A participant was randomly assigned two tasks (corresponding to experimental treatments). Each task entailed a sequence of 20 submissions of manipulated instances of an "ideal" email by the subjects.[1] For each submission the subjects saw an interface similar to the one shown in Figure 6.2.

Each task/treatment included three randomly generated pieces:

1. An "ideal" email instance, visible to the subjects,

[1]The first five submissions were "trials" and were not used towards calculating the final score.

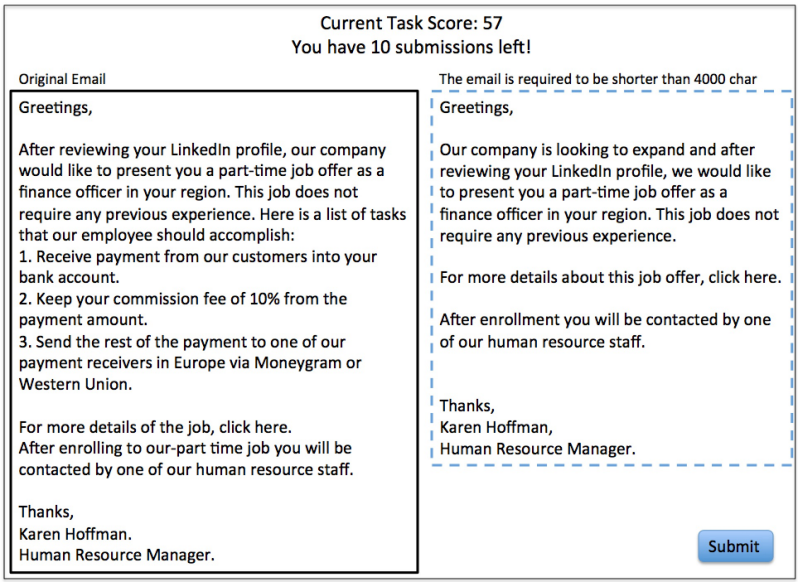


Figure 5.2: Example interface. The left window is the original or "ideal" email. The right window is a free text form for the submission. Once the participant has written the new email (which may involve copying and pasting portions of the original on the left), they click "Submit" to submit it to the system for scoring. In this case, the prior submission bypassed the filter, receiving a score of 57. The participant can make at most 20 submissions. In this example, the participant has made 10 submissions thus far, with 10 more remaining.

2. a classifier filtering submissions, not visible to the subjects, and

3. a scoring function, not visible to the subjects.

An "ideal" email instance corresponds in the evasion model above to the ideal feature vector x; in order to maximize their score, the subjects had to craft an email which was close to this ideal. Ideal instances for the experiments were chosen from 10 pre-selected spam and phishing emails (see the Supplement for details). The actual classifier used in each task was fixed to the model described above. What we varied was whether or not noise was added to a classification output $g(x)$ for a given instance (submission) x. Specifically, in the "noise" treatment the output of the classifier was flipped with 10% probability. The baseline (no noise) treatment, on the other hand, used the classifier as is for all submissions. Finally, the scoring function was chosen uniformly at random between S_1 and S_2 described above. The subjects were not initially told the specifics of the classifier (nor whether or not it included

noise) or of the scoring function (they could request this information after the experiment).

After each submission, a participant received immediate feedback about whether or not the submission was filtered (filtered submissions received zero score), and, if it was not filtered, the score of their last submission, as well as their current best score. For a given submission e, if the cost function (randomly chosen for that task) evaluated to c, the score was calculated as $r(e) = 100 \times e^{-1.5c}$. The best score obtained over all 20 submissions, r_{max}, was used to determine their payment. Specifically, payment for a task was in the range $\$1 - \$5$, with the actual payment computed as $\$1 + 0.04 \times r_{max}$.

Because individual tasks elicited highly divergent performance by the subjects, our reported results use *normalized* scores, $S(t,i) = 90\frac{r(t)-low(i)}{high(i)-low(i)} + 10$, where $high(i)$ and $low(i)$ represents the highest and lowest score among all submissions for task i, respectively.

5.4 Results

5.4.1 Randomization makes evasion more difficult

Since we randomly assigned subjects to randomized vs. non-randomized classifier treatments, we are able to definitively establish the impact that adding a small amount of noise to the filter has on the ability of subjects to evade it. Table 6.1 supports the hypothesis that adding noise to a filter reduces the effectiveness of human subjects to evade it.

Category	S_1	S_2	Overall Average
Noise-Free Filter	24.02	30.06	26.87
Noisy Filter	19.98	20.58	20.29

Table 5.1: Average subject scores for the two scoring functions in the noise-free and noisy filter treatments.

In particular, the overall average for the noise-free filter is over 30% higher than when noise is added. This result is statistically significant ($p < 0.01$). Moreover, the result remains significant for the two scoring functions individually. In addition, we find that the

scoring function S_2 which does not penalize for substitutions tends to yield higher scores for the subjects. This effect is substantial (25% improvement in average normalized score) and significant ($p < 0.01$) in the noise-free filter treatments. Surprisingly, the effect is quite small (3%) and not statistically significant when noise is present. Next we delve deeper in the details of participant behavior to try to shed some light on these results.

The most natural hypothesis into why randomization has a significant impact on the ability to evade the filter is that it makes the task of learning how the filter operates significantly more challenging. The subject behavior bears this out: 77% of submissions were filtered in the noise-free environment, compared to 82% when noise was present (the comparison is significant with $p < 0.01$). Expanding this result by submission (Figure 6.3), we can observe that not only does the noise-free environment appear to promote much faster learning of how to evade a classifier by the subjects, but the relative difference appears to increase with experience (even as there is clear indication of learning to evade in both cases). Interestingly, we did not observe a similar pattern when it came to scores received

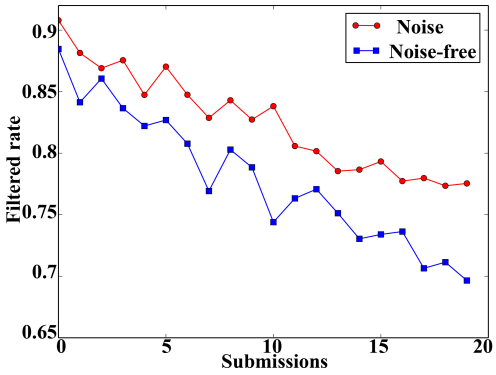


Figure 5.3: Fraction of submissions that were filtered in the noise-free setting (the blue lower line) and when noise was randomly added to filter output (the red higher line).

on non-filtered submissions (filtered submissions, of course, received a score of zero) or in time taken to make a decisions: in both cases, differences were not significant between the noise-free and noisy settings, and there was no clear long-term trend. Thus, for example, there was little evidence that scores earned on non-filtered submissions improved with experience (see the Supplement for details). Similarly, while the first few submissions took

longer than others on average, thereafter the overall trend with experience was minimal (see the Supplement for details). To sum up, it appears that the reduced ability to successfully evade the classifier was the primary effect of randomization on evasion performance.

To further investigate the source of the difficulty that participants faced in the evasion task under randomization, we consider a simple logistic regression model in which the output is the probability that a submission is filtered in the noisy setting. In this model we consider four variables: a binary indicator whether or not a previous submission was flipped (due to noise), the number of prior submission that were flipped, the submission number (higher number indicates a later submission and thereby indicates greater experience), and feature density (fraction of the words in the "ideal" email that are features in the classification-based email filter). The results are shown in Table 6.2. Here x_1 is a binary

	Coefficient	Std Error	z	p-value
x_1	3.11	0.15	21	< 0.01
x_2	0.38	0.025	15	< 0.01
x_3	-0.39	0.021	-18	< 0.01
x_4	7.85	1.11	7.1	< 0.01
constant	-1.53	0.33	-4.7	< 0.01

Table 5.2: Results of the logistic regression model for the probability of a filtered submission.

indicator whether or not a previous submission was flipped (due to noise). x_2 is the number of prior submission that were flipped. x_3 is the submission number. x_4 is the feature density (fraction of the words in the "ideal" email that are features in the classification-based email filter). While all coefficients are highly significant, of particular interest are the first two: whether the prior submission was flipped, and the number of previously flipped submissions. The former increases log-odds by 3.11, or increasing the odds ratio (probability of failed evasion divided by probability of success) by more than 20. Thus, it is quite clear that adding a noise to a particular submission results in a substantial short-term impact on the ability to evade a classifier. In addition, there is a significant longer-term impact: each flip increases the odds ratio of failure to success for any subsequent submission by $\sim$1.5.

Next, we turn to the issue of the impact that a substitution-aware scoring function (S_2) has on performance by considering two micro-level manipulations that participants performed to the "ideal" email as well as subsequent submissions: word additions and deletions. We first look at how these evolved over time.

Figure 6.4 shows that the first several submissions exhibit considerable manipulation, but after the third submission, the numbers of both additions and deletions remains relatively stable, with slightly fewer words added than deleted in each submission except the first few.

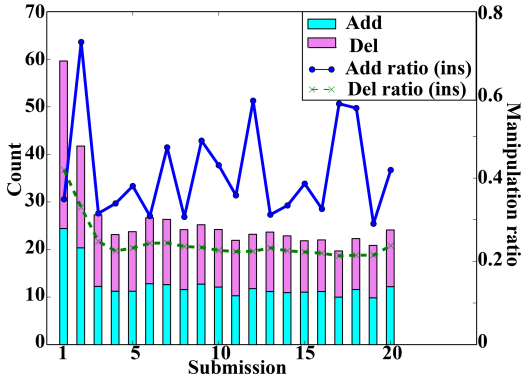


Figure 5.4: Additions and deletions of words to prior submission over time (i.e., over a sequence of submissions). The number of additions is the lower light blue bar, and the heavy blue line represents additions as a fraction of total number of words in the prior submission. The number of deletions is the purple bar at the top, and the dashed green line corresponds to the deletions as a fraction of words in the prior submission.

Overall, we found the number of additions to be higher than deletions ($p < 0.01$). We can also observe that additions comprise a significant fraction of the previous submission, in many cases over 40% of the content. In contrast, the fraction of words deleted was typically around 20%. (The rest of the content is, of course, unchanged from the previous submission). Taking the deletions and additions together, we next compare the total number of *edits* for the noise-free and noisy filter settings. When no noise is added to the filter, the subjects made, on average, nearly 48 edits per submission. In contrast, when noise was present, this number dropped to 45.86 (the difference is significant with $p < 0.01$): somewhat surprisingly, this indicates that the participants were engaged is slightly less

exploration in the noisy-filter treatments, accounting to some extent for the fact that they had significantly more trouble evading the filter in this setting. Moreover, we found that the total fraction of edits which involved substitutions was significantly higher in the noise-free treatment (0.52 vs. 0.47; $p < 0.01$). Again, this points to significantly increased activity and engagement by the subjects in the noise-free treatment (and, perhaps, less confusion about how to effectively manipulate the template). In addition, this helps explain the surprising result that the difference between S_2 and S_1 treatments was much smaller when noise was present: the substitution-aware scoring function S_2 will reward subjects especially for using substitutions in their manipulations, and these were far more prominent under the noise-free treatment.

While substitutions clearly played an important role in subject behavior under all treatments, the composition was somewhat surprising: synonyms accounted for less than 3% of all edits, as compared to character substitutions, which amounted to over 45% of edits. Although some of the character substitutions may have been incidental, rather than deliberate (many words are only a few characters apart), many were clearly deliberate; for example, over 22% of 2-character substitutions cannot be found in the dictionary, suggesting that subjects deliberately misspelled words to evade the classifier.

5.4.2 Positive feedback improves engagement in the task

Next we investigate the extent to which receiving a positive feedback on prior submission impacts human subject performance. We quantify feedback as the difference between the score for prior submission $r(e_{i+1})$ and the score received for the immediately preceding submission $r(e_i)$. We say that feedback is *positive* when this difference is positive ($r(e_{i+1}) - r(e_i) > 0$), and it is *negative* when this difference is negative ($r(e_{i+1}) - r(e_i) < 0$). When these scores are both zero, we say that feedback is *null*, whereas when they are both equal and positive ($r(e_{i+1}) = r(e_i) > 0$), we call it *equal*. We find that receiving positive feedback on prior submission leads subjects to spend *more time* on the following submis-

sion ($p < 0.01$; see Figure 6.5 (a)), and obtain a higher score. In contrast, receiving null or equal feedback leads to less time spent on the following submission. This observation is quite surprising: one would think that particularly null feedback (corresponding to several filtered submissions in a row) would cause the subjects to spend more time contemplating a better evasion strategy, but we see the opposite. Interestingly, our findings are consistent with Mason and Watts [147]. Briefly, Mason and Watts consider a problem of exploring a complex landscape in human subject experiments on a network, where participants could observe what their network neighbors have found. One of their key findings is that when a subject's neighbors find good solutions, the subject engages in significantly *more* exploration of the landscape. Taken together, their findings and ours suggest that positive feedback serves as an important psychological motivator of engagement in a task, which in turn improves performance.

5.4.3 Feature reduction makes adversarial evasion easier

There was significant variability among the 10 tasks (original "ideal" emails) in terms of apparent difficulty of evasion. To understand the source of this variability, we consider the relationship between *feature density*, or the fraction of words in the ideal instance which are used as features in the classifier (filter), and average as well as maximum score for the task.

Figure 6.5 shows that higher feature density leads to a lower score, appearing to make evasion more difficult for the subjects. Table 6.2, which features a logistic regression model of the probability that a submission is filtered, offers stronger evidence: the coefficient corresponding to feature density is again high and significant (and with submission-level granularity we now have much more data to justify this conclusion): it seems clear that increasing feature density makes adversarial evasion far more challenging.

The observation that increased feature density increases difficulty of evasion has important ramifications for the use of machine learning tools in spam/phish filtering tasks specif-

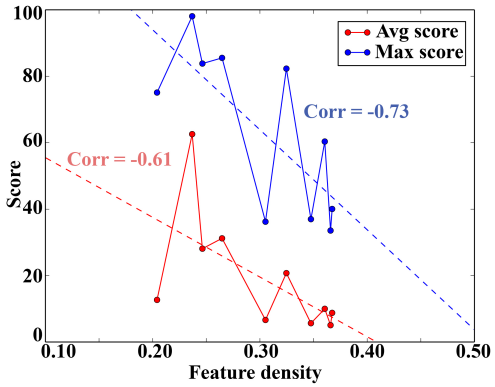


Figure 5.5: The relationship between maximum and average score for each task and the feature density of the corresponding "ideal" email. Feature density is the fraction of words in the ideal email that correspond to features in the classifier (filter).

ically, and intrusion detection more generally. One of the most important "best practice" principles in applied machine learning is feature reduction, or limiting the total number of features used, for example, through the use of regularization or other means of feature selection [139]. Clearly, when we reduce the size of the feature set used for filtering, we also reduce feature coverage of spam instances, that is, we reduce feature density. Our observation here suggests, therefore, that feature reduction can make adversarial evasion easier.

5.4.4 Synthetic model of human evasion behavior

One of our motivations for engaging in human subject evasion experiments was to develop and calibrate a synthetic model of evasion dynamics. We propose the following composite model of behavior:

1. For each feature predict (independently of other features) whether its value is changed, and

2. For each feature word which is predicted to be deleted, predict whether it is substituted for (we assume that substitutions are chosen outside of the feature vector).

For task (1) we develop n independent dynamical models, one corresponding to each binary feature of the classifier, to predict the evolution of the feature vector with the sequence of individual submissions using a Support Vector Machine (SVM) [139]. Features used for this prediction task were taken from a combination of features for the previous two submissions, demographics, as well as feedback, including previous score and whether or not the prior submissions were filtered. For task (2) we developed an independent SVM model for each deleted word predicting occurrence of a substitution. Our models were able to recover underlying behavior with high accuracy in cross-validation (average accuracy was over 97%). However, this in itself is an insufficient criterion for our purposes: a useful dynamic model of behavior should also successfully replicate the experimental findings described earlier, both qualitatively and quantitatively. To exhibit that the model successfully does

Category	S_1	S_2	Overall Average
Noise-Free Filter	28.63	31.61	29.94
Noisy Filter	20.51	21.08	20.75

Table 5.3: Predicted average subject scores for the two scoring functions in the noise-free and noisy filter treatments.

so, Table 6.3 shows the predicted results of our 2x2 treatment (noise-free vs. noisy filter, S_1 vs. S_2 scoring). Comparison to Table 6.1 suggests that the results of the synthetic model closely mirror actual observations of subject behavior in the experiment. In addition, we compare in Figure 6.6 predicted and actual average normalized scores by submission for randomized (noise-free) and non-randomized filters. The predicted and observed average scores match rather closely (nearly perfect correlation and small mean-squared error), and the expected gap between randomized and non-randomized is clearly visible in simulated behavior just as it is in real.

5.5 Summary of Contributions

Machine learning algorithms have come to be used widely in settings which involve inherently adversarial interactions. One of the most important such settings is spam and

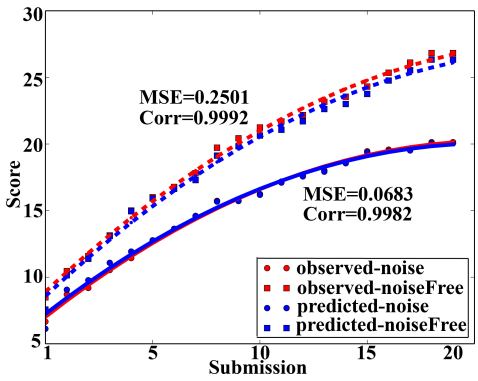


Figure 5.6: Comparison between experimentally observed scores and those based on the synthetic model of behavior as a function of the submission sequence for noisy and noise-free settings.

phishing email filtering, where evasion has been an important and well documented consideration [138, 148]. Our results offer both qualitative practical guidance for the design of robust email filtering systems based on machine learning methods, and explore human behavior in the context of adversarial interaction with a machine learning system. In particular, we can offer several practical pieces of advice: first, filtering systems should embed a small amount of noise in order to increase the level of difficulty for spammers and phishers in designing templates to evade these systems; and second, special care must be taken in training classifiers for such systems to not be overly aggressive in removing "unnecessary" features.

From the perspective of individual behavior, our findings reinforce a surprising finding due to Mason and Watts [147] that observation of good outcomes (we call this positive feedback) increases individual engagement in the task. Indeed, Mason and Watts contextualized this finding entirely in a social network setting, whereas our results suggest that this phenomenon is more fundamental.

Finally, our results have significant bearing on the substantial literature preoccupied with designing high-quality randomization schemes in security [140, 141, 142, 149] and machine learning [114, 113]. In much of this work, randomization schemes are developed under the assumption that the adversary is fully rational. Our experiments demonstrate that

randomization takes on additional importance with a human adversary. In particular, the introduction of noise appears to significantly hamper the ability of the subjects to learn how to evade the classification-based filter. Moreover, the effect of noise is not merely short-term (immediately following the noisy feedback) but has a significant lasting impact on performance well after the perturbation had been introduced.

Chapter 6

ITERATIVE SECURE LEARNING IN ADVERSARIAL ENVIRONMENTS

A number of custom methods have been developed for both adversarial evasion attacks and robust learning. Previous work has taken into account the adversarial strategies from various perspectives. Beyond these theoretical and empirical analysis, in this chapter we unify these work and apply a general practical framework to enhance the robustness of any classifier with respect to any general adversaries, as well as offer the theoretic utility bound for the learner.

Therefore here I introduce the first systematic and general-purpose retraining framework which can: a) boost robustness of an arbitrary learning algorithm, and b) incorporate a broad class of adversarial models. We show that, under natural conditions, the retraining framework minimizes an upper bound on optimal adversarial risk, and show how to extend this result to account for approximations of evasion attacks. We also offer a very general adversarial evasion model and algorithmic framework based on coordinate greedy local search. This is the first such general framework which can be applied for both continuous and discrete-valued feature spaces.

6.1 Overview

Machine learning has been used ubiquitously for a wide variety of security tasks, such as intrusion detection, malware detection, spam filtering, and web search [9, 11, 150, 151, 152]. Traditional machine learning systems, however, do not account for adversarial manipulation. For example, in spam detection, spammers commonly change spam email text to evade filtering. As a consequence, there have been a series of efforts to both model adversarial manipulation of learning, such as evasion and data poisoning attacks [45, 153, 154],

as well as detecting such attacks [11, 123] or enhancing robustness of learning algorithms to these [43, 155, 156, 33, 84, 157]. One of the most general of these, due to Li and Vorobeychik [43], admits evasion attacks modeled through a broad class of optimization problems, giving rise to a Stackelberg game, in which the learner minimizes an adversarial risk function which accounts for optimal attacks on the learner. The main limitation of this approach, however, is scalability: it can solve instances with only 10-30 features. Indeed, most approaches to date also offer solutions that build on specific learning models or algorithms. For example, specific evasion attacks have been developed for linear or convex-inducing classifiers [45, 153, 154], as well as neural networks for continuous feature spaces [21]. Similarly, robust algorithms have typically involved non-trivial modifications of underlying learning algorithms, and either assume a specific attack model or modify a specific algorithm. The more general algorithms that admit a wide array of attack models, on the other hand, have poor scalability.

We propose a very general retraining framework, *RAD*, which can boost evasion robustness of arbitrary learning algorithms using arbitrary evasion attack models. Our first result is to show that *RAD* minimizes an upper bound on optimal adversarial risk; whereas the latter is in general intractable to compute, *RAD* itself is extremely scalable in practice. We develop *RAD* for a more specific, but very broad class of adversarial models, offering a theoretical connection to adversarial risk minimization even when the adversarial model is only approximate. In the process, we offer a simple and very general class of local search algorithms for approximating evasion attacks, which are experimentally quite effective. Perhaps the most appealing aspect of the proposed approach is that it requires no modification of learning algorithms: rather, it can wrap any learning algorithm "out-of-the-box". Our work connects to, and systematizes, several previous approaches which used training with adversarial examples to either evaluate robustness of learning algorithms, or enhance learning robustness. For example, Goodfellow et al. [158] Kantchelian et al. [159] make use of adversarial examples. In the former case, however, these were essentially randomly

chosen. The latter offered an iterative retraining approach more in the spirit of *RAD*, but did not systematically develop or analyze it. Teo et al. [156] do not make an explicit connection to retraining, but suggest equivalence between their general invariance-based approach using column generation and retraining. However, the two are not equivalent, and Teo et al. did not study their relationship formally.

We illustrate the applicability and efficiency of our method on both spam filtering and handwritten digit recognition tasks, where evasion attacks are extremely salient [129, 160]. Figure 7.1 represents how this general retraining framework stays robust and which components within the learning framework it protects.

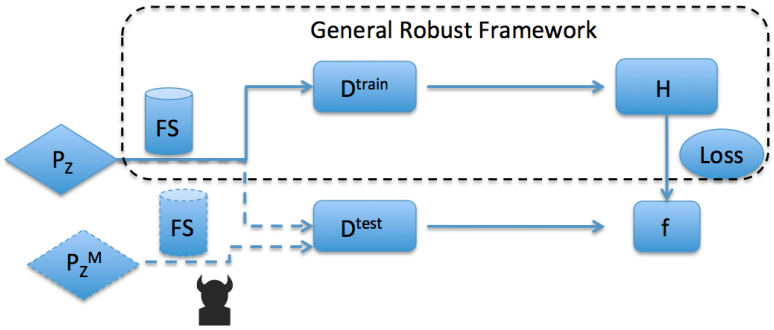


Figure 6.1: General robust retraining framework

6.2 Learning and Evasion Attacks

Let $X \subseteq R^n$ be the feature space, with n the number of features. For a feature vector $x_i \in X$, we let x_{i_j} denote the jth feature. Suppose that the training set (x_i, y_i) is comprised of feature vectors $x_i \in X$ generated according to some unknown distribution $x_i \sim F$, with $y_i \in \{-1, +1\}$ the corresponding binary labels, where -1 means the instance x_i is benign, while $+1$ indicates a malicious instance. The learner aims to learn a classifier with parameters β, $g_\beta : X \to \{-1, +1\}$, to label instances as malicious or benign, using a training data set of labeled instance $D = \{(x_1, y_1), ..., (x_m, y_m)\}$. Let I_{bad} be the subset of datapoints i with $y_i = +1$. We assume that $g_\beta(x) = \text{sgn}(f_\beta(x))$ for some real-valued function $f_\beta(x)$.

Traditionally, machine learning algorithms rely on minimizing regularized empirical risk over training data:

$$\min_{\beta} \mathscr{L}(\beta) \equiv \sum_{i} l(g_\beta(x_i), y_i) + \alpha \|\beta\|_p, \tag{6.1}$$

where $l(\hat{y}, y)$ is the loss associated with predicting $\hat{y}$ when true classification is y. An important issue in adversarial settings is that instances classified as malicious (in our convention, corresponding to $g_\beta(x) = +1$) are associated with malicious agents who subsequently modify such instances in order to evade the classifier (and be classified as benign). Suppose that adversarial evasion behavior is captured by an oracle, $\mathscr{O}(\beta, x)$, which returns, for a given parameter vector β and original feature vector (in the training data) x, an alternative feature vector x'. Since the adversary modifies malicious instances according to this oracle, the resulting effective risk for the defender is no longer captured by Equation 7.1, but must account for adversarial response. Consequently, the defender would seek to minimize the following *adversarial risk* (on training data):

$$\begin{aligned} \min_{\beta} \mathscr{L}_A(\beta; \mathscr{O}) = & \sum_{i: y_i = -1} l(g_\beta(x_i), -1) \\ & + \sum_{i: y_i = +1} l(g_\beta(\mathscr{O}(\beta, x_i), +1) + \alpha \|\beta\|_p^p. \end{aligned} \tag{6.2}$$

The adversarial risk function in Equation 7.2 is extremely general: we make, at the moment, no assumptions on the nature of the attacker oracle, $\mathscr{O}$. This oracle may capture evasion attack models based on minimizing evasion cost [45, 43, 21], or based on actual attacker evasion behavior obtained from experimental data [161].

6.3 Adversarial Learning through Retraining

A number of approaches have been proposed for making learning algorithms more robust to adversarial evasion attacks [44, 43, 155, 156, 84]. However, these approaches

typically suffer from three limitations: 1) they usually assume specific attack models, 2) they require substantial modifications of learning algorithms, and 3) they commonly suffer from significant scalability limitations. For example, a recent, general, adversarial learning algorithm proposed by Li and Vorobeychik [43] makes use of constraint generation, but does not scale beyond 10-30 features.

Recently, retraining with adversarial data has been proposed as a means to increase robustness of learning [158, 159, 156].[1] However, to date such approaches have not been systematic.

We present a new algorithm, *RAD*, for retraining with adversarial data (Algorithm 5) which systematizes some of the prior insights, and enables us to provide a formal connection between retraining with adversarial data, and adversarial risk minimization in the sense of Equation 7.2. The *RAD* algorithm is quite general, and it is not difficult to see that, as

Algorithm 5 RAD: Retraining with ADversarial Examples

Input: training data X
$\bar{\mathscr{X}} \leftarrow X$
$N_i \leftarrow \emptyset \; \forall \; i \in I_{bad}$
repeat
  $new \leftarrow \emptyset$
  for $i \in I_{bad}$ **do**
    $x' = \mathscr{O}(\beta, x_i)$
    if $x' \notin N_i$ **then**
      $new \leftarrow new \cup x'$
    end if
    $N_i \leftarrow N_i \cup x'$
  end for
  $\beta \leftarrow$ Retrain($\bar{\mathscr{X}} \cup_i N_i$)
until $new = \emptyset$
Output: Parameter vector β

described so far, it may never terminate if we include continuous features. However, as the following result attests, it is guaranteed to terminate if the range of $\mathscr{O}$ is a finite set.

Proposition 6.3.1. *Suppose that the range of $\mathscr{O}$ is finite. Then* RAD *will terminate after a*

[1] Indeed, neither Teo et al. [156] nor Kantchelian et al. [159] focus on retraining as a main contribution, but observe its effectiveness.

finite number of iterations.

Proof. Suppose Algorithm 5 does not terminate after iteration k. Then $new \neq \emptyset$, and there is some $x \notin \mathscr{X}$ which is added to the dataset. Since the range of $\mathscr{O}$ is finite, there must be some iteration where there is no x that can be added which is not already in $\mathscr{X}$. Thus, $new = \emptyset$ in that iteration, and the algorithm terminates. □

A special case of this proposition is that the algorithm will always terminate if the feature space X is finite. In practice, to ensure that the algorithm always terminates, we can also impose a strict iteration limit, or check convergence (in terms of how much the parameter vector β changes in successive iterations). In the remainder of the content, however, we assume that there is no fixed iteration limit or convergence check. Instead, we consider what happens if the algorithm does terminate. In particular, define the regularized empirical risk in the last iteration of *RAD* as:

$$\mathscr{L}_N^R(\beta, \mathscr{O}) = \sum_{i \in D \cup N} l(g_\beta(x_i), y_i) + \alpha ||\beta||_p^p, \tag{6.3}$$

where a set $N = \cup_i N_i$ of data points has been added by the algorithm (we omit its dependence on $\mathscr{O}$ to simplify notation). We now characterize the relationship between $\mathscr{L}_N^R(\beta, \mathscr{O})$ and $\mathscr{L}_A^*(\mathscr{O}) = \min_\beta \mathscr{L}_A(\beta, \mathscr{O})$.

Proposition 6.3.2. *$\mathscr{L}_A^*(\mathscr{O}) \leq \mathscr{L}_N^R(\beta, \mathscr{O})$ for all $\beta, \mathscr{O}$.*

Proof. Let $\bar{\beta} \in \arg\min_\beta \mathscr{L}_N^R(\beta, \mathscr{O})$. Consequently, for any β,

$$\begin{aligned}
\mathscr{L}_N^R(\beta, \mathscr{O}) &\geq \mathscr{L}_N^R(\bar{\beta}, \mathscr{O}) \\
&= \sum_{i:y_i=-1} l(g_{\bar{\beta}}(x_i), -1) + \\
&\qquad \sum_{i:y_i=+1} \sum_{j \in N_i \cup x_i} l(g_{\bar{\beta}}(x_i), +1) + \alpha ||\bar{\beta}||_p^p \\
&\geq \sum_{i:y_i=-1} l(g_{\bar{\beta}}(x_i), -1) + \\
&\qquad \sum_{i:y_i=+1} l(g_{\bar{\beta}}(\mathscr{O}(\bar{\beta}, x_i)), +1) + \alpha ||\bar{\beta}||_p^p \\
&\geq \min_\beta \mathscr{L}_A(\beta; \mathscr{O}) = \mathscr{L}_A^*(\mathscr{O}),
\end{aligned}$$

where the second inequality follows because in the last iteration of the algorithm, $new = \emptyset$ (since it must terminate after this iteration), which means that $\mathscr{O}(\beta, x_i) \in N_i$ for all $i \in I_{bad}$. □

In words, retraining, systematized in the *RAD* algorithm, effectively minimizes an upper bound on optimal adversarial risk. This offers a conceptual explanation for the previously observed effectiveness of such algorithms in boosting robustness of learning to adversarial evasion. Formally, however, the result above is extremely limited for several reasons. First, for many adversarial models in prior literature, adversarial evasion is NP-Hard. While some effective approaches exist to compute optimal evasion for specific learning algorithms [159], this is not true in general. Although approximation algorithms for these models exist, using them as oracles in *RAD* is problematic, since actual attackers may compute better solutions, and Proposition 7.3.2 no longer applies. Second, we assume that $\mathscr{O}$ returns a unique result, but when evasion is modeled as optimization, optima need not be unique. Third, there do not exist effective general-purpose adversarial evasion algorithms the use of which in *RAD* would allow reasonable theoretical guarantees. Below, we investigate an important and very general class of adversarial evasion models and associated algorithms

which allow us to obtain practically meaningful guarantees for *RAD*.

6.4 Modeling Attackers

6.4.1 Evasion Attack as Optimization

In prior literature, evasion attacks have almost universally been modeled as optimization problems in which attackers balance the objective of evading the classifier (by changing the label from $+1$ to -1) and the cost of such evasion [45, 43]. Our approach is in the same spirit, but is formally somewhat distinct. In particular, we assume that the adversary has the following two competing objectives: 1) appear as benign as possible to the classifier, and 2) minimize modification cost. Moreover, we assume that the attacker obtains no value from a modification to the original feature vector if the result is still classified as malicious. To formalize, consider an attacker who in the original training data uses a feature vector x_i ($i \in I_{bad}$)). The adversary i is solving the following optimization problem:

$$\min_{x \in X} \min\{0, f(x)\} + c(x, x_i). \tag{6.4}$$

As is typical in related literature, we assume that $c(x, x_i) \geq 0$, $c(x, x_i) = 0$ iff $x = x_i$, and c is strictly increasing in $\|x - x_i\|_2$ and strictly convex in x. Because Problem 7.4 is non-convex, we instead minimize an upper bound:

$$\min_{x} Q(x) \equiv f(x) + c(x, x_i). \tag{6.5}$$

In addition, if $f(x_i) < 0$, we return x_i before solving Problem 7.5. If Problem 7.5 returns an optimal solution x^* with $f(x^*) \geq 0$, we return x_i; otherwise, return x^*. Problem 7.5 has two advantages. First, if $f(x)$ is convex and x real-valued, this is a (strictly) convex optimization problem, has a unique solution, and we can solve it in polynomial time. An important special case is when $f(x) = w^T x$. The second one we formalize in the following

lemma.

Lemma 6.4.1. *Suppose x^* is the optimal solution to Problem 7.4, x_i is suboptimal, and $f(x^*) < 0$. Let $\bar{x}$ be the optimal solution to Problem 7.5. Then $f(\bar{x}) + c(\bar{x}, x_i) = f(x^*) + c(x^*, x_i)$, and $f(\bar{x}) < 0$.*

Proof. If $f(x^*) < 0$, then $\min\{0, f(x^*)\} + c(x^*, x_i) = f(x^*) + c(x^*, x_i)$. By optimality of $\bar{x}$, $f(\bar{x}) + c(\bar{x}, x_i) \leq f(x^*) + c(x^*, x_i)$. Since x_i is suboptimal in Problem 7.4 and c strictly positive in all other cases, $f(x^*) + c(x^*, x_i) < \min\{0, f(x_i)\} + c(x_i, x_i) = 0$. By optimality of x^*, $f(x^*) + c(x^*, x_i) \leq \min\{0, f(\bar{x})\} + c(\bar{x}, x_i) \leq f(\bar{x}) + c(\bar{x}, x_i)$, which implies that $f(\bar{x}) + c(\bar{x}, x_i) = f(x^*) + c(x^*, x_i)$. Consequently, $f(\bar{x}) + c(\bar{x}, x_i) < 0$, and, therefore, $f(\bar{x}) < 0$. □

The following corollary then follows by uniqueness of optimal solutions for strictly convex objective functions over a real vector space.

Corollary 6.4.1. *If $f(x)$ is convex and x continuous, x^* is the optimal solution to Problem 7.4, $\bar{x}$ is the optimal solution to Problem 7.5, and $f(x^*) < 0$, then $\bar{x} = x^*$.*

A direct consequence of this corollary is that when we use Problem 7.5 to approximate Problem 7.4 and this approximation is convex, we always return either the optimal evasion, or x_i if no cost-effective evasion is possible. An oracle $\mathscr{O}$ constructed on this basis will therefore return a unique solution, and supports the theoretical characterization of *RAD* above.

The results above are encouraging, but many learning problems do not feature a convex $f(x)$, or a continuous feature space. Next, we consider several general algorithms for adversarial evasion.

6.4.2 Coordinate Greedy

We propose a very general local search framework, *CoordinateGreedy (CG)* (Algorithm 6 for approximating optimal attacker evasion. The high-level idea is to iteratively

Algorithm 6 CoordinateGreedy(*CG*): $\mathscr{O}(\beta, x)$

Input: Parameter vector β, malicious instance x
Set $k \leftarrow 0$ and let $x^0 \leftarrow x$
repeat
    Randomly choose index $i_k \in \{1, 2, ..., n\}$
    $x^{k+1} \leftarrow$GreedyImprove(i_k)
    $k \leftarrow k + 1$
until $\frac{\ln Q(x^k)}{\ln Q(x^{k-1})} \leq \varepsilon$
if $f(x^k) \geq 0$ **then**
    $x^k \leftarrow x$
end if
Output: Adversarially optimal instance x^k.

choose a feature, and greedily update this feature to incrementally improve the attacker's utility (as defined by Problem 7.5). In general, this algorithm will only converge to a locally optimal solution. We therefore propose a version with random restarts: run *CG* from L random starting points in feature space. As long as a global optimum has a basin of attraction with positive Lebesgue measure, or the feature space is finite, this process will asymptotically converge to a globally optimal solution as we increase the number of random restarts. Thus, as we increase the number of random restarts, we expect to increase the frequency that we actual return the global optimum. Let p_L denote the probability that the oracle based on coordinate greedy with L random restarts returns a suboptimal solution to Problem 7.5. The next result generalizes the bound on *RAD* to allow for this, restricting however that the risk function which we bound from above uses the $0/1$ loss. Let $\mathscr{L}^*_{A,01}(\mathscr{O})$ correspond to the total adversarial risk in Equation 7.2, where the loss function $l(g_\beta(x), y)$ is the $0/1$ loss. Suppose that $\mathscr{O}_L$ uses coordinate greedy with L random restarts.

Proposition 6.4.2. *Let* $B = |I_{bad}|$.

$$\mathscr{L}^*_{A,01}(\mathscr{O}) \leq \mathscr{L}^R_N(\beta, \mathscr{O}_L) + \delta(p)$$

with probability at least $1-p$*, where*

$$\delta(p) = B\left(p_L + \frac{\sqrt{\log^2 p - 8Bp_l \log p} - \log p}{2B}\right),$$

and $\mathscr{L}_N^R(\beta, \mathscr{O}_L)$ *uses any loss function* $l(g_\beta(x), y)$ *which is an upper bound on the* $0/1$ *loss.*

Proof. Let $\bar{\beta} \in \arg\min_\beta \mathscr{L}_N^R(\beta, \mathscr{O}_L)$. Consequently, for any β,

$$\begin{aligned}
\mathscr{L}^*_{A,01}(\mathscr{O}_L) &= \min_\beta \mathscr{L}_{A,01}(\beta; \mathscr{O}_L) \\
&\leq \sum_{i:y_i=-1} l_{01}(g_{\bar{\beta}}(x_i), -1) + \\
&\qquad \sum_{i:y_i=+1} l_{01}(g_{\bar{\beta}}(\mathscr{O}(\bar{\beta}, x_i)), +1) + \alpha||\bar{\beta}||_p^p.
\end{aligned}$$

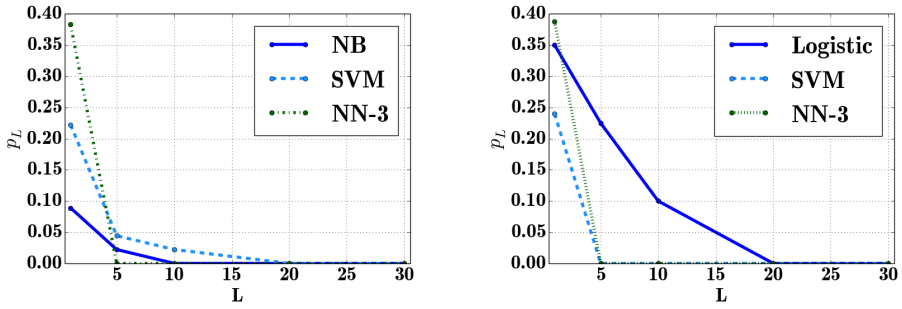
Now,

$$\begin{aligned}
&\sum_{i:y_i=+1} l_{01}(g_{\bar{\beta}}(\mathscr{O}(\bar{\beta}, x_i)), +1) \leq \\
&\qquad \sum_{i:y_i=+1} l_{01}(g_{\bar{\beta}}(\mathscr{O}_L(\bar{\beta}, x_i)), +1) + \delta(p)
\end{aligned}$$

with probability at least $1-p$, where $\delta(p) = Bp_L + \frac{\sqrt{\log^2 p - 8Bp_l \log p} - \log p}{2}$, by the Chernoff bound, and Lemma 7.4.1, which assures that an optimal solution to Problem 7.5 can only over-estimate mistakes. Moreover,

$$\begin{aligned}
&\sum_{i:y_i=+1} l_{01}(g_{\bar{\beta}}(\mathscr{O}_L(\bar{\beta}, x_i)), +1) \leq \\
&\qquad \sum_{i:y_i=+1} \sum_{j \in N_i} l(g_{\bar{\beta}}(\mathscr{O}_L(\bar{\beta}, x_i)), +1),
\end{aligned}$$

since $\mathscr{O}_L(\bar{\beta}, x_i) \in N_i$ for all i by construction, and l is an upper bound on l_{01}. Putting everything together, we get the desired result. □



(a) (b)

Figure 6.2: The convergence of p_L based on different number of starting points for (a) Binary, (b) Continuous feature space.

Figure 7.2 demonstrates that $p_L \to 0$ quite rapidly for an array of learning algorithms, and for either discrete or continuous features, as we increase the number of restarts L. Consequently, in practice retraining with coordinate greedy nearly minimizes an upper bound on minimal adversarial risk using a 0/1 loss with relatively few restarts of the approximate attacker oracle.

6.4.2.1 Continuous Feature Space

For continuous feature space, we assume that both $f(x)$ and $c(x,\cdot)$ are differentiable in x, and propose using the *coordinate descent* algorithm, which is a special case of coordinate greedy, where the GreedyImprove step is:

$$x^{k+1} \leftarrow x^k - \tau_k e_{i_k} \frac{\partial Q(x^k)}{\partial x^k_{i_k}},$$

where τ_k is the step size and e_{i_k} the direction of i_kth coordinate. Henceforth, let the origial adversarial instance x_i be given; we then simplify cost function to be only a function of x, denoted $c(x)$. If the function $f(x)$ is convex and differentiable, our coordinate descent based algorithm 6 can always find the global optima which is the attacker best response x^* [162], and Proposition 7.3.2 applies, by Corollary 7.4.1. If $f(x)$ is not convex, then coordinate descent will only converge to a local optimum.

Now, $\frac{\partial Q(x^k)}{\partial x^k_{i_k}} = \frac{\partial f(x^k)}{\partial x^k_{i_k}} + \frac{\partial c(x^k)}{\partial x^k_{i_k}}$. Fixing a coordinate $j = i_k$, we derive the partial derivative of $f(x)$ with respect to coordinate j for several common learning algorithms, and derive the derivative of $c(x)$ for several natural cost functions.

6.4.2.1.1 Kernel SVM Consider a general Kernel SVM decision function, for which $f(x) = \sum_i a_i y_i K(x_i, x)$, where x_i, y_i are the support training data points, a_i the associated dual SVM parameters, and $K(\cdot,\cdot)$ a kernel function. Then, $\frac{\partial f(x)}{\partial x_j} = \sum_i a_i y_i \frac{\partial K(x_i,x)}{\partial x_j}$. For a linear Kernel, $\frac{\partial K(x_i,x)}{\partial x_j} = x_{i_j}$. For a polynomial Kernal, $\frac{\partial K(x_i,x)}{\partial x_j} = d(c + x_i \cdot x)^{d-1} x_{i_j}$. For an RBF Kernel, $\frac{\partial K(x_i,x)}{\partial x_j} = \frac{1}{\sigma^2} K(x_i,x)(x_{i_j} - x_j)$.

6.4.2.1.2 Logistic Regression Given a logistic regression model, with $f(x) = (1 + e^{-w^T x})^{-1}$. Then, $\frac{\partial f(x)}{\partial x_j} = w_j f(x)(1 - f(x))$.

6.4.2.1.3 Neural Network For the neural network with sigmoid function as the activation function, if there are three hidden layers,

$$
\begin{aligned}
g(x) &= (1 + e^{-h_3(x)})^{-1}, \\
h_3(x) &= \sum_{t=1}^{d_3} w_{3t} \delta_{3t}(x) + b_3, \\
\delta_{3t}(x) &= (1 + e^{-h_{2t}(x)})^{-1}, \\
h_{2t}(x) &= \sum_{j=1}^{d_2} w_{2j} \delta_{2j}(x) + b_2, \\
\delta_{2j}(x) &= (1 + e^{-h_{1j}(x)})^{-1}, \\
h_{1j}(x) &= \sum_{p=1}^{d_1} w_{1p} \delta_{1p}(x) + b_1, \\
\delta_{1p}(x) &= (1 + e^{-h_{0p}(x)})^{-1}, \\
h_{0p}(x) &= \sum_{l=1}^{n} w_{1l} x_l + b_0,
\end{aligned}
$$

Therefore we have

$$\frac{\partial f(x)}{\partial x_j} = g(x)(1-g(x)) \sum_{t=1}^{d_3} w_{j_3 t} \delta_{j_3 t} (1-\delta_{j_3 t}) \\ \sum_{k=1}^{d_2} w_{j_2 k} \delta_{j_2 j} (1-\delta_{j_2 k}) \sum_{p=1}^{d_1} w_{j_1 p} \delta_{j_1 p} (1-\delta_{j_1 p}) w_{j_0 i}. \tag{6.6}$$

6.4.2.1.4 Quadratic Cost A simple and natural cost function is a quadratic (l_2) cost, $c(x,x_i) = \frac{\lambda}{2}\|x-x_i\|_2^2$. In this case, $\frac{\partial c(x)}{\partial x_j} = \lambda(x_j - x_{ij})$.

6.4.2.1.5 Exponential Cost Below, we make use of an exponential cost, which is also quite natural: options become exponentially less desirable to an attacker as they are more distant from their ideal attack. We use the following cost function:

$$c(x,x_i) = \exp\left(\lambda(\sum_j (x_j - x_{ij})^2 + 1)^{1/2}\right).$$

Then,

$$\frac{\partial c(x)}{\partial x_j} = \frac{\lambda c(x,x_i)(x_j - x_{ij})}{(\sum_j (x_j - x_{ij})^2 + 1)^{1/2}}.$$

6.4.3 Discrete Feature Space

In the case of discrete feature space, GreedyImprove step of *CG* can simply enumerate all options for feature j, and choose the one most improving the objective.

6.4.4 Attacks as Constrained Optimization

A variation on the attack models above is when the attacker is solving the following constrained optimization problem:

$$\min_x \min\{0, f(x)\} \tag{6.7a}$$

$$\text{s.t.}: \quad c(x, x_i) \leq B \tag{6.7b}$$

for some cost budget constraint B. While this problem is, again, non-convex, we can instead minimize the convex upper bound, $f(x)$, as before, if we assume that $f(x)$ is convex. In this case, if the feature space is continuous, the problem can be solved optimally using standard convex optimization methods [163]. If the feature space is binary and $f(x)$ is linear or convex-inducing, algorithms proposed by Lowd and Meek [45] and Nelson et al. [154].

6.5 Multi-class classification

Discussion so far dealt entirely with binary classification. We now observe that extending it to multi-class problems is quite direct. Specifically, while previously the attacker aimed to make an instance classified as $+1$ (malicious) into a benign instance (-1), for a general label set Y, we can define a malicious set $M \subset Y$ and a target set $T \subset Y$, with $M \cap T = \emptyset$, where every entity represented by a feature vector x with a label $y \in M$ aims to transform x so that its label is changed to T. In this setting, let $g(x) = \arg\max_{y \in Y} f(x, y)$. We can then use the following empirical risk function:

$$\sum_{i: y_i \notin M} l(g_\beta(x_i), y_i) + \sum_{i: y_i \in M} l(g_\beta(\mathscr{O}(\beta, x_i)), y_i) + \lambda ||\beta||_p, \tag{6.8}$$

where $\mathscr{O}$ aims to transform instances x_i so that $g_\beta(\mathscr{O}(\beta, x_i)) \in T$. The relaxed version of the adversarial problem can then be generalized to

$$\min_{x,y \in T} -f(x,y) + c(x,x_i).$$

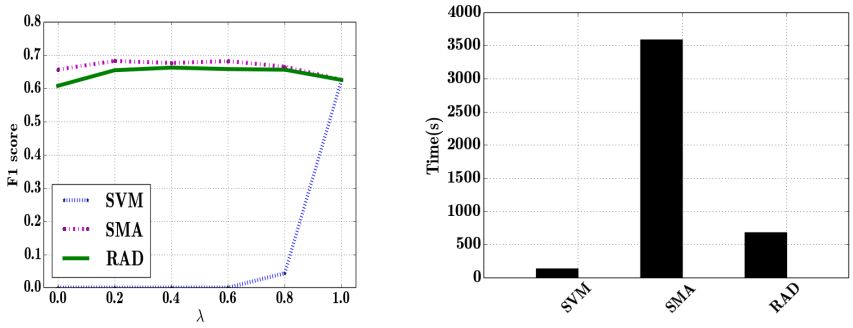
For a finite target set T, this problem is equivalent to taking the best solution of a finite collection of problems identical to Problem 7.5.

6.6 Results

The results above suggest that the proposed systematic retraining algorithm is likely to be effective at increasing resilience to adversarial evasion. We now offer an extensive experimental evaluation of this. Throughout, we make use of the exponential cost model for the attacker described above, where λ is a parameter which determines the relative importance of the cost term. In particular, a high λ implies a higher penalty for changing the malicious feature vector, and, consequently, weaker attacks. Additionally, we simulated attacks using Problem 7.5 formulation.

6.6.1 Comparison to Optimal

The first comparison we draw is to a recent algorithm, *SMA*, which minimizes l_1-regularized adversarial risk function (7.2) using the hinge loss function. Specifically, *SMA* formulates the problem as a large mixed-integer linear program which it solves using constraint generation [43]. The main limitation of SMA is scalability. Because retraining methods use out-of-the-box learning tools, it is considerably more scalable.

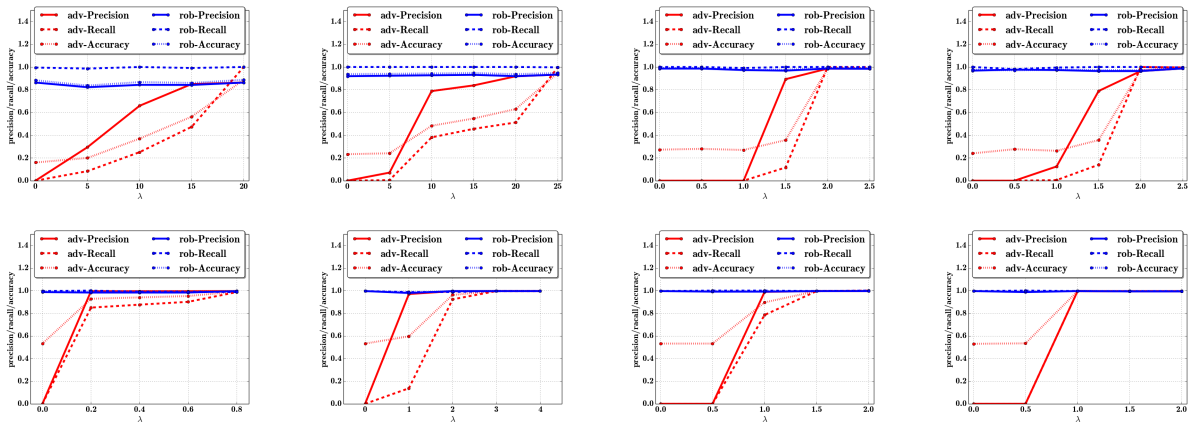


(a) (b)

Figure 6.3: Comparison between *RAD* and *SMA* based on the Enron dataset with 30 binary features. (a) The F_1 score of different algorithms corresponding to various λ; (b) the average runtime for each algorithm.

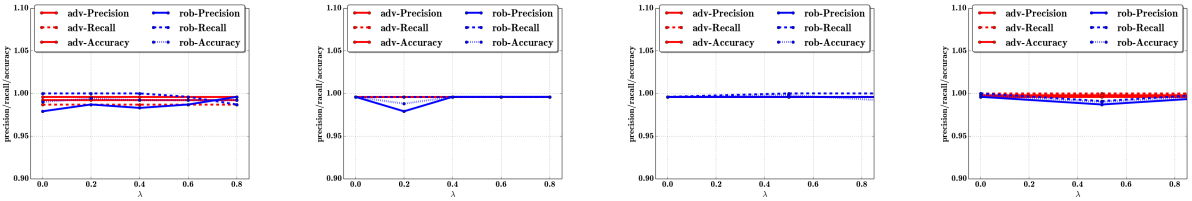
We compared *SMA* and *RAD* (with the same adversarial model) using Enron data [129] since *SMA* was also developed for the constrained-optimization attacks (Problem (7.7)). As Figure 7.3(a) demonstrates, retraining solutions of *RAD* are nearly as good as *SMA*, particularly for a non-trivial adversarial cost sensitive λ. In contrast, a baseline implementation of SVM is significantly more fragile to evasion attacks. However, the runtime comparison for these algorithms in figure 7.3(b) shows that *RAD* is much more efficient than *SMA* due to its neat retraining solution. In addition, Figure 7.3(a) also reveals a surprising phenomenon observed by Kantchelian et al. [159] as well: performance actually improves based on certain adversarial cost sensitivity compared with adversary-free case where high λ value is applied. What this shows is that adding certain adversarial instances actually improves classification robustness to non-adversarial data as well, likely because it makes the learner significantly more robust to noise in the data.

Below, we evaluate the effectiveness of retraining in significantly boosting robustness of learning to evasion.



(a) (b) (c) (d)

Figure 6.4: Performance of baseline (*adv*-) and *RAD* (*rob*-) as a function of cost sensitivity λ for Enron (top) and MNIST (bottom) datasets with continuous features testing on adversarial instances. (a) logistic regression, (b) SVM, (c) 1-layer NN, (d) 3-layer NN.



(a) (b) (c) (d)

Figure 6.5: Performance of baseline (*adv*-) and *RAD* (*rob*-) as a function of cost sensitivity λ for MNIST dataset with continuous features testing on non-adversarial instances. (a) logistic regression, (b) SVM, (c) 1-layer NN, (d) 3-layer NN.

6.6.2 Continuous Feature Space

In this section we use the Enron dataset [129] and MNIST [160] dataset to evaluate the robustness of three common algorithms in their standard implementation, and in *RAD*: logistic regression, SVM (using a linear kernel), and a neural network (NN) with 1 and 3 hidden layers. In Enron data, features correspond to relative word frequencies. 2000 features were used for the Enron and 784 for MNIST datasets. Throughout, we use precision, recall, and accuracy as metrics.

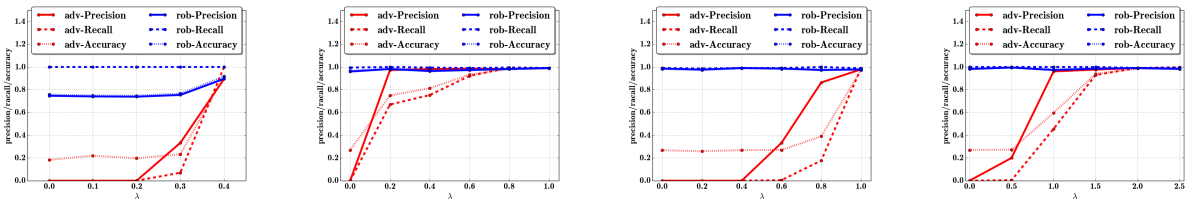
Figure 7.4(a) shows the performance of logistic regression, with and without retraining, on Enron and MNIST. The increased robustness of *RAD* is immediately evident: perfor-

Figure 6.6: Example modification of digit images (MNIST data) as λ decreases (left-to-right) for logistic regression, SVM, 1-layer NN, and 3-layer NN (rows 1-4 respectively).

mance of *RAD* is essentially independent of λ on all three measures, and substantially exceeds baseline algorithm performance for small λ. Interestingly, we observe that the baseline algorithms are significantly more fragile to evasion attacks on Enron data compard to MNIST: benign and malicious classes seem far easier to separate on the latter than the former. This qualitative comparison between the Enron and MNIST datasets is consistent for other classification methods as well (SVM, NN). These results also illustrate that the neural-network classifiers, in their baseline implementation, are significantly more robust to evasion attacks than the (generalized) linear classifiers (logistic regression and SVM): even with a relatively small attack cost attacks become ineffective relatively quickly, and the differences between the performance on Enron and MNIST data are far smaller. Throughout, however, *RAD* significantly improves robustness to evasion, maintaining extremely high accuracy, precision, and recall essentially independently of λ, dataset, and algorithm used.

In order to explore whether *RAD* would sacrifice accuracy when no adversary is present, Figure 7.5 shows the performance of the baseline algorithms and *RAD* on a test dataset sans evasions. Surprisingly, *RAD* is never significantly worse, and in some cases better than non-adversarial baselines: adding malicious instances appears to increase overall generalization ability. This is also consistent with the observation by Kantchelian et al. [159].

In Figure 7.6 we visualize the relative vulnerability of the different classifiers, as well as effectiveness of our general-purpose evasion methods based on coordinate greedy. Each



(a) (b) (c) (d)

Figure 6.7: Performance of baseline (*adv-*) and *RAD* (*rob-*) implementations of (a) Naive Bayes, (b) logistic regression, (c) SVM, and (d) 3-layer NN, using binary features testing on adversarial instances.

row corresponds to a classifier, and moving right within a row represents decreasing λ (allowing attacks to make more substantial modifications to the image in an effort to evade correct classification). We can observe that NN classifiers require more substantial changes to the images to evade, ultimately making these entirely unlike the original. In contrast, logistic regression is quite vulnerable: the digit remains largely recognizable even after evasion attacks.

6.6.3 Discrete Feature Space

Considering now data sets with binary features, we use the Enron data with a bag-of-words feature representation, for a total of 2000 features. We compare Naive Bayes (NB), logistic regression, SVM, and a 3-layer neural network (results for 1-layer NN are similar). Our comparison involves both the baseline, and *RAD* implementations of these, using the same metrics as above.

Figure 7.7 confirms the effectiveness of *RAD*: every algorithm is substantially more robust to evasion with retraining, compared to baseline implementation. Most of the algorithms can obtain extremely high accuracy on this data with the bag-of-words feature representation. However, a 3-layer neural network is now *less* robust than the other algorithms, unlike in the experiments with continuous features. Indeed, Goodfellow et al. [158] similarly observe the relative fragility of NN to evasion attacks.

6.6.4 Multi-class Classification

To evaluate the effectiveness of *RAD*, and resilience of baseline algorithms, in multi-class classification settings, we use the MNIST dataset and aim to correctly identify digits based on their images. Our comparison involves SVM and 3-layer neural network (results for NN-1 are similar). We use $M = \{1,4\}$ as the malicious class (that is, instances corresponding to digits 1 and 4 are malicious), and $T = \{2,7\}$ is the set of benign labels (what malicious instances wish to be classified as). The results, shown in Figure 7.8 are

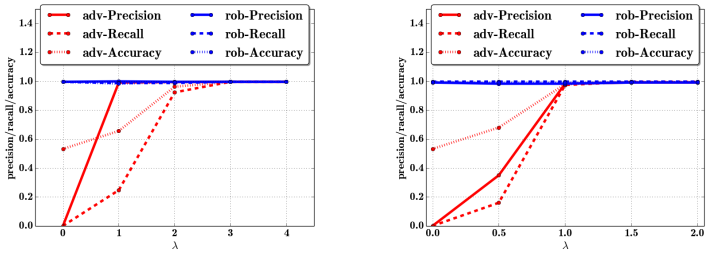


Figure 6.8: Performance of baseline (*adv-*) and RAD (*rob-*) implementations of (a)multi-class SVM and (b) multi-class 3-layer NN, using MNIST dataset testing on adversarial instances.

largely consistent with our previous observations: both SVM and 3-layer NN perform well when retrained with *RAD*, with near-perfect accuracy despite adversarial evasion attempts. Moreover, *RAD* significantly boosts robustness to evasion, particularly when λ is small (adversary who is not very sensitive to evasion costs). Figure 7.9 offers a visual demon-

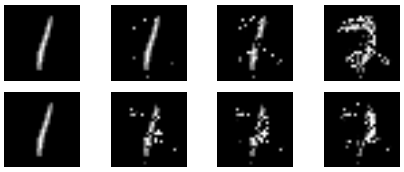


Figure 6.9: Visualization of modification attacks with decreasing the cost sensitivity parameter λ (from left to right), to change 1 to the set $\{2,7\}$. The rows correspond to SVM and 3-layer NN, respectively.

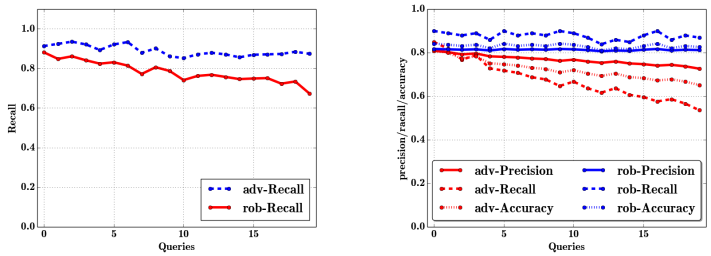
stration of the relative effectiveness of attacks on the baseline implementation of SVM and 1- and 3-layer neural networks. Here, we can observe that a significant change is required

to evade the linear SVM, with the digit having to nearly resemble a 2 after modification. In contrast, significantly less noise is added to the neural network in effecting evasion.

6.6.5 Oracles based on Human Evasion Behavior

Our final set of experiments evaluate *RAD* just for the SVM classifier in the context of human evasion behavior *in human subject experiments*. The data for this evaluation was obtained from the human subject experiment by Ke et al. [161] in which subjects were tasked with the goal of evading an SVM-based spam filter, manipulating 10 spam/phishing email instances in the process. In these experiments, Ke et al. used machine learning to develop a model of human subject evasion behavior. We now adopt this model as the evasion oracle, $\mathcal{O}$, injected in our *RAD* retraining framework, executing the synthetic model for 0-10 iterations to obtain evasion examples.

Figure 7.10(a) shows the recall results for the dataset of 10 malicious emails (the classifiers are trained on Enron data, but evaluated on these 10 emails, including evasion attacks). Figure 7.10(b) shows the classifier performance for the enron dataset by applying the synthetic adversarial model as the oracle for both evasion modification and *RAD*. We can make



(a) (b)

Figure 6.10: *RAD* (*rob-*) and baseline SVM (*adv-*) performance based on human subject behavior data over 20 queries, (a) using experimental data with actual human subject experiment submissions, (b) using Enron data and a synthetic model of human evader.

two high-level observations. First, notice that human adversaries appear significantly less powerful in evading the classifier than the automated optimization-based attacks we previously considered. This is a testament to both the effectiveness of our general-purpose

adversarial evaluation approach, and the likelihood that such automated attacks likely significantly overestimate adversarial evasion risk in many settings. Nevertheless, we can observe that the synthetic model used in *RAD* leads to a significantly more robust classifier. Moreover, as our evaluation used actual evasions, while the synthetic model was used only in training the classifier as a part of *RAD*, this experiment suggests that the synthetic model can be relatively effective in modeling behavior of human adversaries. Figure 7.10(b) performs a more systematic study using the synthetic model of adversarial behavior on the Enron dataset. The findings are consistent with those only considering the 10 spam instances: retraining significantly boosts robustness to evasion, with classifier effectiveness essentially independent of the number of queries made by the oracle.

6.7 Summary of Contributions

Here we provide a general-purpose systematic retraining algorithm against evasion attacks of classifiers for arbitrary oracle-based evasion models. We first demonstrated that this algorithm effectively minimizes an upper bound on optimal adversarial risk, which is typically extremely difficult to compute (indeed, no approach exists for minimizing adversarial loss for an arbitrary evasion oracle). Experimentally, we showed that the performance of our retraining approach is nearly indistinguishable from optimal, whereas scalability is dramatically improved: indeed, with *RAD*, we are able to easily scale the approach to thousands of features, whereas a state-of-the-art adversarial risk optimization method can only scale to 15-30 features. We generalize our results to show that a probabilistic upper bound on minimal adversarial loss can be obtained even when the oracle is computed approximately by leveraging random restarts, and an empirical evaluation which confirms that the resulting bound relaxation is tight in practice.

We also offer a general-purpose framework for optimization-based oracles using variations of coordinate greedy algorithm on both discrete and continuous feature spaces. Our experiments demonstrate that our adversarial oracle approach is extremely effective in cor-

rupting the baseline learning algorithms. On the other hand, extensive experiments also show that the use of our retraining methods significantly boosts robustness of algorithms to evasion. Indeed, retrained algorithms become nearly insensitive to adversarial evasion attacks, at the same time maintaining extremely good learning performance on data overall. Perhaps the most significant strength of the proposed approach is that it can make use of arbitrary learning algorithms essentially "out-of-the-box", and effectively and quickly boost their robustness, in contrast to most prior adversarial learning methods which were algorithm-specific.

In summary, the contributions of this general robust framework are as following:

1. *RAD*, a novel systematic framework for adversarial retraining,
2. analysis of the relationship between *RAD* and optimal empirical adversarial risk,
3. extension of the analysis to account for approximate adversarial evasion, within a specific broad class of adversarial models,
4. extensive experimental evaluation of *RAD* and the adversarial evasion model.

Chapter 7

SCALABLE OPTIMIZATION OF RANDOMIZED OPERATIONAL DECISIONS

Besides the optimal robust learning algorithm based on certain attack model, it is demanded to apply different learning algorithms to corresponding problems instead of solving an universal optimization problem. Therefore, in this chapter we study a more general way towards *secure learning* by taking the output of any probabilistic learning algorithm and applying the optimization approach to provide more robust solutions against different adversarial strategies. The literature on adversarial machine learning aims to develop learning algorithms which are robust to such adversarial evasion, but exhibits two significant limitations: a) failure to account for operational constraints and b) a restriction that decisions are deterministic. To overcome these limitations, here we will introduce a conceptual separation between learning, used to infer attacker *preferences*, and operational decisions, which account for adversarial evasion, enforce operational constraints, and naturally admit randomization. Our approach gives rise to an intractably large linear program. To overcome scalability limitations, we introduce a novel method for estimating a compact parity basis representation for the operational decision function. Additionally, we develop an iterative constraint generation approach which embeds adversary's best response calculation, to arrive at a scalable algorithm for computing near-optimal randomized operational decisions.

7.1 Overview

Success of machine learning across a variety of domains has naturally led to its adoption as a tool in security settings, including intrusion detection, biometric identity recognition, and spam filtering. Unlike traditional uses of machine learning, however, these domains involve an *adversary*, who is likely to adapt to the use of such techniques, po-

tentially reducing their effectiveness. Of particular interest in many such application domains is *adversarial classification*, or the task of determining whether a given input (email, system access, user behavior) is benign or "normal", or malicious (a phishing email or a system compromise). In such settings, we start with a training data set of labeled instances $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, where x_i are feature vectors (e.g., whether or not specific spam/phish indicators are present in an email) and y_i are labels, which we can code as 0 corresponding to benign and 1 to malicious instances. This data set is used to train a classifier, h, that would presumably predict whether an arbitrary unseen instance x is malicious. The phenomenon of *adversarial evasion* puts a damper on this seemingly clean solution: if an adversary wishes, say, to send an email with features x, but $h(x)$ classifies it as malicious, an intelligent attacker would attempt to choose another email, corresponding to x', which would be classified as benign, and achieve the same, or nearly the same, ends.

The literature on *adversarial machine learning* tackles the problem of adversarial evasion in two ways: first, by trying to understand its feasibility and effectiveness [44, 164, 115, 165, 128], and second, by attempting to design machine learning algorithms which account for, and are robust to, evasion [44, 115, 85, 166, 84, 148, 167].

Past literature on algorithm design for adversarial classification suffers from two important limitations. First, previous approaches make no attempt to account for resource constraints involved in *operationalizing* the algorithms: in particular, it is the false positives, rather than false negatives, which are critical to adoption of intrusion detection systems, in large part because overabundance of "alerts" makes such a system operationally unusable [168]. Second, there is, to date, no principled way of embedding randomization into adversarial classification, even though stochasticity in defense is often highly effective in security [140, 169, 170]. Indeed, the use of randomization in adversarial classification has previously been suggested [171], but the proposed approach is ad hoc, simply adding "random noise" to the classifier output.

We address both of these limitations by rethinking the conceptual model of adversarial

classification. Specifically, we separate the task of *learning*, which uses training data to *predict attack preferences*, and the task of *operational decisions*, which uses the resulting predictor, together with an evasion model, in computing optimal *randomized* operational policy that *explicitly abides by operational constraints*. The intuition for this separation is that the training data can be interpreted as *revealed preferences* of the attackers, in the sense that the attacks captured by it can be viewed as "ideal" attack vectors at that point in time. As an indirect consequence, our model enables one to use off-the-shelf machine learning packages, allowing progress in machine learning and adversarial decision making to be decoupled. On the technical side, we present a natural generalization of a commonly used evasion model (see, e.g., [165]) to randomized classification settings. We show that computing an optimal evasion is NP-Hard, but also exhibit an optimal branch-and-bound search method and two polynomial-time approximation algorithms, one with worst-case performance guarantees, and both shown to be "near-optimal" in experiments. On the operational side, we introduce a linear programming (LP) formulation for computing optimal randomized classification. While the baseline LP involves an exponential number of variables and constraints, we propose a collection of techniques which make use of a Fourier representation of Boolean functions [172], as well as constraint generation, to arrive at scalable approximation. The general structure of this work is shown as in Figure 8.1.

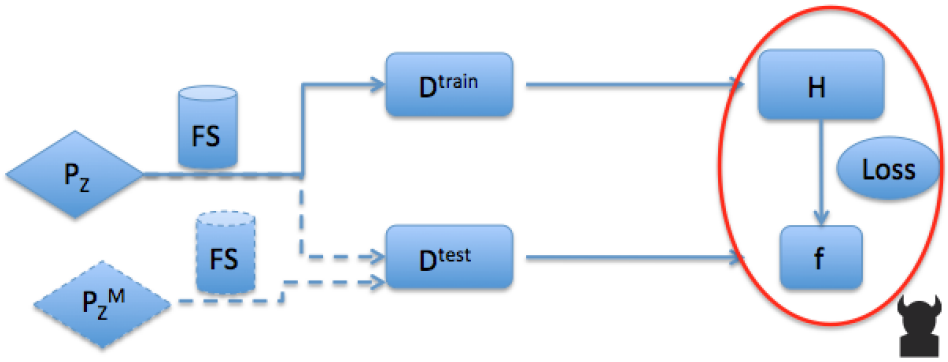


Figure 7.1: General idea of dynamic operational decisions

7.2 Model

We consider the adversarial binary classification problem over an input space $\mathscr{X}$, where each input feature vector $x \in \mathscr{X}$ can be categorized (labeled) as benign or malicious. The defender $\mathscr{D}$, collects a data set of labeled instances, $\mathscr{I} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, which we assume to accurately represent the current distribution of input instances and corresponding categories. $\mathscr{D}$ then applies an algorithm of choice, such as Naive Bayes, to obtain a probabilistic classifier which assigns to an arbitrary input x vector a probability $p(x)$ that it is generated by a malicious actor *assuming such an actor does not change their behavior*. In traditional applications, one would then use a threshold, θ, and classify an instance x as malicious if $p(x) \geq \theta$, and benign otherwise. This decision (and the choice of the threshold) are often motivated by overall tolerance for false positives, as well as operational considerations, for example, to ensure that the number of alerts does not exceed what can reasonably be inspected by security professionals. To consider operational decisions in general, as well as allow for randomization, we introduce a function $q(x, p(\cdot)) \in [0,1]$ which prescribes a possibly randomized operational decision (e.g., the probability of filtering an email or manually investigating an observed network access pattern) for an instance x given a prediction $p(x)$. To simplify notation, we simply use $q(x)$ where $p(\cdot)$ is clear from context. Throughout, we assume that features are binary, a common case in adversarial classification settings (e.g., features could correspond to specific words or phrases being present in email, or specific sequences of system calls executed).

We model adversarial classification as a Stackelberg game between a defender and a population of attackers. In this game, the defender $\mathscr{D}$ moves first, choosing $q(\cdot)$. Next, the attackers learn $q(\cdot)$ (for example, through probing), and each attacker subsequently chooses an input vector x (e.g., a phishing email) to maximize their expected return (a combination of bypassing defensive countermeasures and achieving a desired outcome). Our assumption that the operational policy $q(\cdot)$ is known to attackers reflects threats that have significant time and/or resources to probe and respond to defensive measures, a feature characteristic

of advanced cyber criminals [173].

Attacker Model

We interpret the data set $\mathscr{I}$ and the resulting predictions $p(x)$, as representing *revealed preferences* of a sample of attackers, that is, their preference for input vectors x. Our rationale is that if an attacker preferred some other input x', this attacker would have chosen x' instead of x in $\mathscr{I}$. Consequently, $p(x)$ can be interpreted as an "ideal" attack, if only it were to succeed in bypassing defensive measures. If $q(x)$ is sufficiently close to 1, x is likely to fail, and the attacker will have an incentive to *evade* by choosing another instance x'. When decisions $q(x)$ are deterministic, a common approach in related literature is to assume that the attacker will find x' which is closest to x (in some distance metric, such as l_1 norm) of all alternatives classified as benign [174, 165, 166, 84, 148]. We now offer a natural generalization of this model to account for randomized $q(x)$. Specifically, if the attacker with a preference for x chooses an alternative attack vector x', we model his utility from successfully bypassing defenses as $V(x)Q(x,x')$, where $Q(x,x') = e^{-\delta||x-x'||}$, with $||\cdot||$ a norm (we use Hamming distance), $V(x)$ the value of the attack, and δ the importance of being close to the preferred x. The full utility function of an attacker with preference x for choosing another input x' when the defense strategy is q is then

$$\mu(x,x';q) = V(x)Q(x,x')(1-q(x')), \tag{7.1}$$

since $1-q(\cdot)$ is the probability that the attacker successfully bypasses the defensive action.

While the above attacker model admits considerable generality, we assume that attackers fall into two classes: adaptive, as described above, and static, corresponding to the limiting case of $\delta \to \infty$. Let $v_t(x;q)$ be the value function of an attacker with type t and preference for x, when the defender chooses a policy q. $v_t(x;q)$ represents the maximum utility that the attacker with type t can achieve given q. For a static attacker, the value function is $v_S(x;q) = V(x)(1-q(x))$, that is, a static attacker always uses his preferred input x,

and receives his corresponding value for it whenever the defender does not take action upon observing x. For an adaptive attacker, the value function is $v_A(x;q) = \max_{x'} \mu(x,x';q)$, that is, the maximum utility that the attacker obtains from using an *arbitrary* input x'. Finally, let P_A be the probability that an arbitrary malicious input was generated by an adaptive adversary; the probability that the adversary was static is then $P_S = 1 - P_A$.

Defender Model

A natural goal for the defender is to maximize expected value of benign traffic that is classified as benign, less the expected losses due to attacks that successfully bypass the operator. To formalize, we assume that the defender gains a positive value $G(x)$ from a benign input x only if it is not inspected. In the case of email traffic, this is certainly sensible if our action is to filter a suspected email. More generally, inspection can be a lengthy process, in which case we can interpret $G(x)$ as the value of time lost if x is, in fact, benign, but is carefully screened before it can have its beneficial impact. We define the defender's utility function $U_{\mathscr{D}}(q,p)$ as follows:

$$U_{\mathscr{D}}(q,p) = \mathbb{E}_x\left[(1-q(x))G(x)(1-p(x)) - \right.$$
$$\left. p(x)(P_S v_S(x;q) + P_A v_A(x;q))\right].$$

To interpret the defender's utility function, let us first rewrite it for a special case when $V(x) = G(x) = 1$ and $P_S = 1$, reducing the utility function to $\mathbb{E}_x[(1-q(x))(1-p(x)) - p(x)(1-q(x))]$. Since $p(x)$ is constant, this is equivalent to minimizing

$$\mathbb{E}_x[q(x)(1-p(x)) + p(x)(1-q(x))],$$

which is just the *expected misclassification error*.

The final aspect of our model is a resource constraint on the defender. Sommer and Paxson [168] identify the cost of false positives and the gap between the output of machine learning algorithms and its use in operational decisions as two of the crucial gaps that pre-

vent widespread use of machine learning in network intrusion detection. In our framework, $G(x)$ quantifies the loss of value due to false positives. We handle the hard constraint on defensive resources by introducing a budget constraint that our solution inspects at most a fraction c of events, on average.

Model Analysis A natural sanity check that our formulation is reasonable is that the solution corresponds to intuition when there is no budget constraint or adaptive adversary. We now show that in this case, the policy $q(x)$ which uses a simple threshold on $p(x)$ (as commonly done) is, in fact optimal.

Proposition 7.2.1. *Suppose that $P_A = 0$ and $c = 1$ (i.e., no budget constraint). Then the optimal policy is*

$$q(\vec{x}) = \begin{cases} 1 & \text{if } p(\vec{x}) \geq \frac{G(\vec{x})}{G(\vec{x})+V(\vec{x})} \\ 0 & \text{o.w.} \end{cases}$$

Proof. Since we consider only static adversaries and there is no budget constraint, the objective becomes

$$\max_{\vec{q}} \sum_{\vec{x} \in \mathscr{X}} \left[(1 - q(\vec{x}))G(\vec{x})(1 - p(\vec{x})) - p(\vec{x})v_S(\vec{x}) \right],$$

and the only remaining constraint is that $q(\vec{x}) \in [0,1]$ for all $\vec{x}$. Since now the objective function is entirely decoupled for each $\vec{x}$, we can optimize each $q(\vec{x})$ in isolation for each $\vec{x} \in \mathscr{X}$. Rewriting, maximizing the objective for a given $\vec{x}$ is equivalent to minimizing $q(\vec{x})[G(\vec{x}) - p(\vec{x})(G(\vec{x}) + V(\vec{x}))]$. Whenever the right multiplicand is negative, the quantity is minimized when $q(\vec{x}) = 1$, and when it is positive, the quantity is minimized when $q(\vec{x}) = 0$. Since $p(\vec{x}) \geq \frac{G(\vec{x})}{G(\vec{x})+V(\vec{x})}$ implies that the right multiplicand is negative (more accurately, non-positive), the result follows. □

While traditional approaches threshold an odds ratio (or log-odds) rather than the probability $p(x)$, the two are, in fact equivalent. To see this, let us consider the generalized (cost-sensitive) threshold on odds ratio used by the Dalvi et al. [174] model. In their no-

tation, $U_{\mathscr{C}}(+,+)$, $U_{\mathscr{C}}(+,-)$, $U_{\mathscr{C}}(-,+)$, and $U_{\mathscr{C}}(-,-)$ denote the utility of the defender (classifier) when he correctly identifies a malicious input, incorrectly identifies a benign input, incorrectly identifies a malicious input, and correctly identifies a benign input, respectively. In our setting, we have $U_{\mathscr{C}}(+,+) = 0$ (i.e., no loss), $U_{\mathscr{C}}(+,-) = 0$ (and capture the costs of false positives as operational constraints instead), $U_{\mathscr{C}}(-,+) = -V(x)$, and $U_{\mathscr{C}}(-,-) = G(x)$ (note that we augment the utility functions to depend on input vector x). The odds-ratio test used by Dalvi et al. therefore checks

$$\frac{p(x)}{1-p(x)} \geq \frac{U_{\mathscr{C}}(-,-) - U_{\mathscr{C}}(+,-)}{U_{\mathscr{C}}(+,+) - U_{\mathscr{C}}(-,+)} = \frac{G(x)}{V(x)}. \tag{7.2}$$

and it is easy to verify that inequality 8.2 is equivalent to the threshold test in Proposition 8.2.1.

Consider now a more general setting where $P_A = 0$, but now with a budget constraint. In this context, we now show that the optimal policy is to first set $q(x) = 0$ for all x with $p(x)$ below the threshold described in Proposition 8.2.1, then rank the remainder in descending order of $p(x)$, and assign $q(x) = 1$ in this order until the budget is exhausted.

Proposition 7.2.2. *Suppose that $P_A = 0$. Then the optimal policy is to let $q(x) = 0$ for all x with*

$$p(x) < \frac{G(x)}{G(x) + V(x)}.$$

Rank the remaining x in descending order of $p(x)$ and set $q(x) = 1$ until the budget is exhausted, leaving the remaining budget to the next instance x, and setting $q(x) = 0$ for the rest.

Proof. The LP can be rewritten so as to minimize

$$\sum_{\vec{x}} q(\vec{x})[G(\vec{x}) - p(\vec{x})(G(\vec{x}) + V(\vec{x}))]$$

subject to the budget constraint. By the same argument as above, whenever $p(\vec{x})$ is below the threshold, the optimal $q(\vec{x}) = 0$. Removing the corresponding $\vec{x}$ from the objective, we

obtain a special knapsack problem in which the above greedy solution is optimal, since the coefficient on the budget constraint is 1. □

In a nutshell, Proposition 8.2.2 suggests that whenever the budget constraint binds, we should simply inspect the highest priority items. Therefore, randomization becomes important only when there is an adversary actively responding to our inspection efforts.

7.3 Optimal Randomized Operational Use of Classification

Given the Stackelberg game model of strategic interactions between a defender armed with a classifier, and an attacker attempting to evade it we now develop an algorithmic approach for solving it. We begin by using a sample average approximation of the defender's utility function $U_{\mathscr{D}}$ (e.g., using instances in the training data), denoting it $\hat{U}_{\mathscr{D}}$. Using $\hat{U}_{\mathscr{D}}$ as the objective, we can maximize it using the following linear program (LP):

$$\max_{q} \quad \hat{U}_{\mathscr{D}}(q,p) \tag{7.3a}$$

$$\text{s.t.:} \quad 0 \leq q(x) \leq 1 \qquad \forall\, x \in \mathscr{X} \tag{7.3b}$$

$$v_A(x;q) \geq \mu(x,x';q) \qquad \forall\, x,x' \in \mathscr{X} \tag{7.3c}$$

$$v_S(x;q) = V(x)(1-q(x)) \qquad \forall\, x \in \mathscr{X} \tag{7.3d}$$

$$\mathbb{E}_x[q(x)] \leq c, \tag{7.3e}$$

where constraint 8.3c computes the attacker's best response (optimal evasion of q).

Scaling Up

The linear program 8.3 is not a practical solution approach for two reasons: a) $q(x)$ must be defined over the entire feature space $\mathscr{X}$, and b) the set of constraints is quadratic in $|\mathscr{X}|$. Since with n features $|\mathscr{X}| = 2^n$, this LP is a non-starter.

Our first step towards addressing the scalability issue is to represent $q(x)$ using a set of normalised basis functions, $\{\phi_j(x)\}$, where $q(x) = \sum_j \alpha_j \phi_j(x)$. This allows us to focus on optimizing α_j, a potentially tractable proposition if the set of basis functions is small. With

this representation, the LP now takes the following form (to simplify exposition below, we assume that $P_A = 1$; generalization is direct):

$$\min_{\alpha \geq 0} \sum_j \alpha_j \mathbb{E}[G(x)\phi_j(x)(1-p(x))] + \mathbb{E}[V(x)p(x)Q(x,\alpha)] \tag{7.4a}$$

$$\text{s.t} : Q(x,\alpha) \geq e^{-\delta||x-x'||}(1 - \sum_i \alpha_j \phi_j(x')) \;\; \forall x, x' \in \mathscr{X} \tag{7.4b}$$

$$\sum_j \alpha_j \mathbb{E}[\phi_j(x)] \leq c \tag{7.4c}$$

$$\sum_j \alpha_j \leq 1. \tag{7.4d}$$

While we can reduce the number of variables in the optimization problem using a basis representation ϕ, we still retain the intractably large set of inequalities which compute the attacker's best response. To address this issue, suppose that we have an oracle $\mathscr{O}(x;q)$ which can efficiently compute a best response x' to a strategy q for an attacker with an ideal attack x. Armed with this oracle, we propose a constraint generation aproach, termed Adaptive Adversary based Scalable classification (AAS), to iteratively compute an (approximately) optimal operational decision function q (Algorithm 7 below).

Algorithm 7 AAS(X)

```
φ =ConstructBasis()
X̄ ← X
q ← MASTER(X̄)
while true do
  for x ∈ X_bad do
    x' = O(x;q)
    X̄ ← X̄ ∪ x'
  end for
  if All x' ∈ X̄ then
    {If no new x' generated}
    return q
  end if
  q ← MASTER(X̄)
end while
```

The input to the AAS algorithm (Algorithm 7) is the feature matrix X in the training

data, with X_{bad} denoting this feature matrix restricted to "bad" (malicious) instances. At the core of Algorithm 7 is the MASTER linear program which computes an attacker's (approximate) best response using the modified LP 8.4, but using only a small subset of all feature vectors as alternative attacks, which we denote by $\bar{\mathscr{X}}$. The algorithm begins with $\bar{\mathscr{X}}$ initialized to only include feature vectors in the training data X. The first step is to compute an optimal solution, q, with adversarial evasion restricted to X. Then, iteratively, we compute each attacker's best response x' to the current solution, q, adding it to $\bar{\mathscr{X}}$ (the preferences of each attacker are parameterized by the attacks they executed in the original training data), rerun the MASTER linear program to recompute q, and repeat. The process is terminated when we cannot generate any new constraints (i.e., the available constraints already include best responses for all attackers). The following result is a direct consequence of a) finiteness of feature space, and b) the fact that at termination the attacker is playing an actual best response to the computed strategy q.

Theorem 7.3.1. *The AAS algorithm computes an optimal solution q given a fixed basis ϕ in finite time.*

The approach described so far in principle addresses the scalability issues, but leaves two key questions unanswered: 1) how do we construct the basis ϕ, a problem which is of critical importance to good quality approximation (the ConstructBasis() function in Algorithm 7), and 2) how do we compute the attacker's best response to q, represented above by an oracle $\mathscr{O}(x,q)$. We tackle these in turn.

Basis Construction

Our basis representation relies on harmonic (Fourier) analysis of Boolean functions [172, 175]. In particular, it is known that every Boolean function $f : \{0,1\}^n \to \mathbb{R}$ can be uniquely represented as $f(x) = \sum_{S \in B_S} \hat{f}_S \chi_S(x)$, where $\chi_S(x) = (-1)^{S^T x}$ is a parity function on a given basis $S \in \{0,1\}^n$, B_S is the set containing all the bais S, and the corresponding Fourier coefficients can be computed as $\hat{f}_S = \mathbb{E}_x[f(x)\chi_S(x)]$ [176, 175]. Our goal will be to approximate $q(x)$ using a Fourier basis. Our core task is to compute a set of basis functions to be sub-

sequently used in optimizing $q(x)$. The first step is to uniformly randomly select K feature vectors $\vec{x_k}$. Then use a traditional learning algorithm, say Naive Bayes, to obtain the $p(x)$ vector and solve the linear program 8.3 to compute $q(x)$ restricted to these. We can now use the same set of feature vectors to approximate a Fourier coefficient of this $q(x)$ for an arbitrary basis S as $t = \frac{1}{m}\sum_{i=1}^{m} q(x^i)\chi_S(x^i)$. We can use this expression to compute a basis set S with the largest Fourier coefficient using the following integer linear program:

$$\max_{S} \quad \frac{1}{K}\sum_{k=1}^{K} q(x^k) r_S^k \tag{7.5a}$$

$$s.t.: \quad S^T x^k = 2y^k + h^k \tag{7.5b}$$

$$r_S^k = 1 - 2h^k \tag{7.5c}$$

$$y^k \in Z, h^k \in \{0,1\}, S \in \{0,1\}^n \tag{7.5d}$$

Our basis generation algorithm solves this program iteratively, each time adding a constraint that rules out a previously generated basis, until the optimal solution is zero. Each basis is optimized within limited time and then we collect the set of optimized basis functions B_S that are corresponding to the largest Fourier coefficients. To consider the largest negative Fourier coefficients, we simply change Program 8.5 to be minimization. We found, however, that negative Fourier coefficients were rare in our problem instances.

Computing Adversary's Best Response

The constraint generation algorithm AAS described above presumes the existance of an oracle $\mathcal{O}(x;q)$ which computes (or approximates) an optimal evasion of q (we call this a *best response* to q) for an attacker that would prefer to use a feature vector x. We now address this problem in detail. Note that since $V(x)$ is fixed in the attacker's evasion problem (because x is fixed), it can be ignored.

We begin by addressing the computational complexity of computing an optimal evasion. Informally, given an arbitrary set of bases ϕ and the adversary's preference feature vector x, the attacker wishes to modify as few features as possible to obtain a binary vector x' that minimizes $q(x')$. To make the analysis cleaner, we compute the bases ϕ_j as mapping

to $\{0,1\}$, where $\phi_j(x) = \frac{1}{2}(\chi_{S_j}(x)+1)$. A formal decision problem faced by the attacker is whether there exist a feature vector x' satisfying the following constraints:

$$\sum_j \alpha_j \phi_j(x') \leq \lambda \tag{7.6a}$$

$$\|x - x'\| \leq k, \tag{7.6b}$$

where λ and k are fixed given thresholds. This problem, which we call *EVASION*, can be shown to be computationally hard by reducing it from 3DM.

Theorem 7.3.2. *EVASION is NP-complete.*

Proof. This adversary evasion problem is in NP, as we can non-deterministically pick $a \leq k$ features and verify if $q(\vec{x'}) \leq \lambda$.

We prove that the problem is NP-hard via a reduction from 3-dimensinal matching (3DM). For an arbitrary instance of 3DM, W, Y, and Z are finite, disjoint sets with the same number of d elements. T is a subset of $W \times Y \times Z$, which means T consists of triples (w,y,z) such that $w \in W, y \in Y$, and $z \in Z$. $M \subseteq T$ ($|M| = d$) is a 3-dimensional matching if for any two distinct triples $(w_1,y_1,z_1) \in M$ and $(w_2,y_2,z_2) \in M$, $w_1 \neq w_2$, $y_1 \neq y_2$, and $z_1 \neq z_2$.

Each triple $(w_i,y_i,z_i) \in T$ corresponds to one feature, which controls a set of basis $(s_{w_i}, s_{d+y_i}, s_{2d+z_i})$. There are $n = |T|$ features and $m = |W| + |Y| + |Z| = 3d$ basises, which forms the basis matrix as the figure 8.2 below. Each elements within the matrix $b_{ji} = 1$ denotes that the *jth* basis is controlled by the *ith* feature; otherwise 0. As each feature controls exactly one basis from each part, we have for any feature $i(1 \leq i \leq n)$ and basis $j(1 \leq j \leq m)$, $\sum_{j=1}^{d} b_{ji} = 1$, $\sum_{d+1}^{2d} b_{ji} = 1$, $\sum_{2d+1}^{3d} b_{ji} = 1$, $(d = \frac{1}{3}m)$. Let $k = d$, $\lambda = q(x) - 3d/D$, $(D \geq 3d)$, $\Delta = q(\vec{x}) - q(\vec{x'})$. If $q(x') \leq \lambda$, we have $\Delta = q(\vec{x}) - q(\vec{xx'}) \geq 3d/D$. Let $\alpha_1 = \alpha_2 = ... = \alpha_m = \frac{1}{2D}$, and x is a vector with all 0. Therefore $\phi_j(x) = \alpha_j(-1)^{s_j x} = \frac{1}{2D}$ for $1 \leq j \leq m$. Consequentially, let $x^{l'}$ denotes the modified instance x', which only differs in feature l with x. If $b_{hl} = 1$, the corresponding basis function would flip the sigh, thus

$\phi_h(x^{l\prime}) = \alpha_l(-1)^{s_h x^{l\prime}} = -\frac{1}{2D}$. Suppose there are J bases that have been flipped the sign,

$$\begin{aligned}\Delta =& q(\vec{x}) - q(\vec{x'}) = \sum_{j=1}^{m} \alpha_j(-1)^{s_j x} - \sum_{j=1}^{m} \alpha_j(-1)^{s_j x'} \\ =& \left(\sum_{j\in J} \alpha_j(-1)^{s_j x} + \sum_{j\in S\setminus J} \alpha_j(-1)^{s_j x}\right) - \\ & \left(\sum_{j\in J} \alpha_j(-1)^{s_j x'} + \sum_{j\in S\setminus J} \alpha_j(-1)^{s_j x'}\right).\end{aligned}$$

As $\sum_{j\in S\setminus J} \alpha_j(-1)^{s_j x} = \sum_{j\in S\setminus J} \alpha_j(-1)^{s_j x'}$, $\Delta = \frac{1}{2D}|J| - (-\frac{1}{2D})|J| = \frac{|J|}{D}$, which means the decrement of $q(x)$ equals to the number of basises that would flip the sign divided by D. It is easy to see how this construction can be accomplished in polynomial time. Therefore, suppose

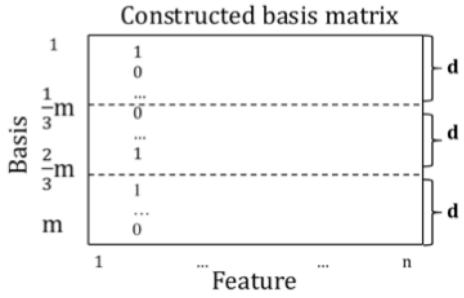


Figure 7.2: Illustration for the problem construction

there are $a \leq k$ features that can be modified in x to satisfy that $q(\vec{x'}) \leq \lambda$. It follows that $\Delta = q(\vec{x}) - q(\vec{x'}) \geq 3d/D$. Additionally, as each feature only control 3 basises, the total number of basis that would flip the sign is $\Delta = q(\vec{x}) - q(\vec{x'}) \leq 3a/D \leq 3k/D = 3d/D$. It derives that $\Delta = 3d/D$, which means there is no overlap between selected basis. Accordingly, subset M ($|M| = d$) is chosen and each triple $(w_i, y_i, z_i) \in M$ corresponds to the set of controlled basises by feature i. Therefore the total number of elements within the selected subsets in M satisfies $|W| + |Y| + |Z| = \Delta \cdot D = 3d$. So any two selected distinct triples $(w_1, y_1, z_1) \in M$ and $(w_2, y_2, z_2) \in M$, $w_1 \neq w_2$, $y_1 \neq y_2$, and $z_1 \neq z_2$. This means if there is a solution for the adversary evasion problem, there exists a 3-dimensional matching.

Conversely, suppose M is a 3DM. The d selected exclusive triples correspond to $k = d$

specific feature, each of which controls 3 basis. As all the triples are non-overlapped, there are $3d$ different responding basises that would flip the sign, which means $q(\vec{x'}) = q(\vec{x}) - \Delta = q(\vec{x}) - 3d/D = \lambda$. Therefore, the adversary evasion problem can be solved if and only if a 3DM exists. □

Since adversarial evasion is NP-Hard, it is natural to develop an approximation algorithm to solve the following derived optimization problem:

$$\min_{x'} \sum_j \alpha_j \phi_j(x') \tag{7.7a}$$

$$\text{s.t.:} \quad ||x - x'|| \leq k \tag{7.7b}$$

Define $\Delta(x') = q(x) - q(x') = \sum_j \alpha_j(\phi_j(x) - \phi_j(x'))$, so that our objective can equivalently be stated as maximizing $\Delta(x')$ so that at most k features in x are modified. Let Δ^* be the optimal solution to this problem. We present Algorithm 9 to compute x' which yields a provably near-optimal solution.

Algorithm 8 ApproxEvasion(F, q, k, ε)

```
n ← |F|
D_0 ← {(∅, 0)} {tuple d_i = (feaSet, value) ∈ D}
G = GenFeaGroup(F, S)
l ← 0
for i ← 1 to |G| do
    for j ← 1 to |g_i| do
        l ← l + 1 {merge two tuple-lists by d_i.value}
        D_l ← MergeTuple(D_{l-1}, AddFea(D_{l-1}, f_ij, k))
    end for
    D_l ← Trim(D_l, ε/2n)
    remove elements from D_l that d_l.value > q(x)
end for
let d* correspond to the maximum d.value in D_n
return d*
```

Theorem 7.3.3. *Suppose that the number of inputs in any basis is bounded by a constant c. Then* ApproxEvasion *(Algorithm 9) computes a solution x' to problem 8.7 which achieves $\hat{\Delta} \geq \frac{\Delta^*}{1+\varepsilon}$, where $\hat{\Delta} = \Delta(x')$ in time poly$(n, \frac{1}{\varepsilon}, 2^c)$.*

Proof. The operations of $Trim$ and removing from D_l every member that is greater than $q(\vec{x})$ maintain the property that every element of D_l meets our decreasing requirement. For every element d_i in D_i that the corresponding retrieved value is at most $q(x)$, there exists an element $d_k \in D_i$ such that $\frac{Retrieve(d_i)}{(1+\varepsilon/2n)^i} \leq Retrieve(d_k) \leq Retrieve(d_i)$. This must hold for the optimal Δ^*, therefore there exists an element $d \in D_n$ that $\frac{\Delta^*}{(1+\varepsilon/2n)^n} \leq Retrieve(d) \leq \Delta^*$. Thus $\frac{\Delta^*}{Retrieve(d)} \leq (1+\frac{\varepsilon}{2n})^n$. As this inequality must also hold for $\hat{\Delta}$, $\frac{\Delta^*}{\hat{\Delta}} \leq (1+\frac{\varepsilon}{2n})^n$. Since $\lim_{n\to\infty}(1+\varepsilon/2n)^n = e^{\varepsilon/2}$ and $\frac{d}{dn}(1+\varepsilon/2n)^n > 0$, the function $(1+\varepsilon/2n)^n$ increases with n and we have $(1+\varepsilon/2n)^n \leq e^{\varepsilon/2} \leq 1+\varepsilon/2+(\varepsilon/2)^2 \leq 1+\varepsilon$.

Therefore, $\hat{\Delta} \geq \frac{\Delta^*}{1+\varepsilon}$.

Next we show that it is a polynomial-time approximation scheme based on a restrictive feature group size c, which is the maximum size of each feature group obtained from Algorithm 10. To analyze the run time, we need to derive the bound on the length of D_i. After trimming between groups of features, successive elements d and d' of D_i must have the relationship $d'/d > 1+\varepsilon/2n$. That is, they must differ by a factor of at least $1+\varepsilon/2n$. Each list, therefore, constraints the value 0, possibly value $\delta > 0$, which is a small number less than the minimal α value, and up to $\lfloor \log_{1+\varepsilon/2n} \frac{q(x)}{\delta} \rfloor$. Therefore we can derive that the number of elements in each list D_i is at most

$$2^c\left(\log_{1+\varepsilon/2n}\frac{q(x)}{\delta}+2\right) = 2^c\left(\frac{\ln\frac{q(x)}{\delta}}{\ln(1+\varepsilon/2n)}+2\right) \tag{7.8}$$

$$\leq 2^c\left(\frac{2n(1+\varepsilon/2n)\ln\frac{q(x)}{\delta}}{\varepsilon}+2\right) \tag{7.9}$$

$$< 2^c\left(\frac{3n\ln\frac{q(x)}{\delta}}{\varepsilon}+2\right). \tag{7.10}$$

Therefore, this bound of the list length is polynomial in the size of the input n when $c \leq \log_2 n$. Since the running time of *ApproxAdversaryEvasion* is polynomial in the lengths of the D_i, we conclude that there is a polynomial-time approximation scheme ($O(n \cdot 2^c)$) with

respect to the restricted feature group size as c ($c \leq \log_2 n$). □

Algorithm 9 returns the approximate solution of maximum Δ to obtain a minimal $q(\vec{x'})$ by modifying less or equal to k features.

Algorithm 9 $ApproxAdversaryEvasion(F, q(\vec{x}), \varepsilon, k)$

$n \leftarrow |F|$
$D_0 \leftarrow \{(\emptyset, 0)\}$ {tuple $d_i = (feaSet, value) \in D$}
$G = GenFeaGroup(F, S)$
$l \leftarrow 0$
for $i \leftarrow 1$ to $|G|$ **do**
  for $j \leftarrow 1$ to $|g_i|$ **do**
    $l \leftarrow l + 1$ {merge two tuple-lists by $d_i.value$}
    $D_l \leftarrow MergeTuple(D_{l-1}, AddFea(D_{l-1}, f_{ij}, k))$
  end for
  $D_l \leftarrow Trim(D_l, \varepsilon/2n)$
  remove elements from D_l that $d_l.value > q(\vec{x})$
end for
let d^* correspond to the maximum $d.value$ in D_n
return d^*

As the length of D_i can be 2^i, which makes the merge algorithm take exponential time, here we employ the Algorithm 12 to trim the list length. The idea is that if some combi-

Algorithm 10 $GenFeaGroup(F, S)$

$G \leftarrow \emptyset$
$n \leftarrow |F|$
$m \leftarrow |S|$
for $j \leftarrow 1$ to m **do**
  for $i \leftarrow 1$ to n **do**
    $g_j \leftarrow \emptyset$
    if $s_{ji} = 1$ **then**
      $g_j \leftarrow g_j \cup f_i$
    end if
  end for
  $G \leftarrow G \cup g_j$
end for
$G \leftarrow DisjointSet(G)$ {convert G to disjoint-sets}
return G

nation of features make the decrease of $q(\vec{x})$ similar, then only one combination should be

kept. This means that with a trimming parameter δ, for any element d_i removed from D_i, there is an element d_j that approximates d_i, that is,

$\frac{Retrieve(d_i)}{1+\delta} \leq Retrieve(d_j) \leq Retrieve(d_i)$.

However, this $Trim$ action can only be done for features that have no common basises to avoid missing qualified feature combination. Therefore, Algorithm 10 is applied to group the features that need to be added as a whole before $Trim$; and algorithm 11 helps to form different feature combinations and guarantee only less or equal to k features are considered.

Algorithm 11 $AddFea(D, f, k)$

$m \leftarrow |D|$
$D' \leftarrow \emptyset$
for $i \leftarrow 1$ to m **do**
  if $size(i.set \cup f) \leq k$ **then**
    $t_i'.set \leftarrow t_i.set \cup f$
    $t_i'.value \leftarrow Retrieve(t_i'.set)$
    insert t_i' into ordered D' by $t_i'.value$
  end if
end for
return D'

Algorithm 12 $Trim(D, \varepsilon)$

$m \leftarrow |D|$
$D' \leftarrow d_1$
$last \leftarrow d_1.value$
for i $\leftarrow$ 2 to m **do**
  if $d_i.value > last \cdot (1+\varepsilon)$ **then**
    append d_i onto the end of D'
    $last \leftarrow d_i.value$
  end if
end for
return D'

Finally, for each feature combination we would use the algorithm 13 to obtain the corresponding value based on the chosen bases. Our goal is to find the feature combination that reduce the most from $q(\vec{x})$ by flipping fewer features, which means the strategy x' can have a higher chance to pass the classifier after less modification on the original "ideal" instance x.

Algorithm 13 $Retrieve(d)$

$w \leftarrow \emptyset$
for $f_i \in d$ **do**
  $w \leftarrow w \oplus w_{f_i}$ $\{w_{f_i}$ is basis set controled by $f_i\}$
end for
$v \leftarrow \sum_{s_j \in w} -2\alpha_{xj}$ $\{\alpha_{xj}$ is the actual value in $x\}$
return v

While the complete algorithm and proof are in the extended version, below we offer some intuition. The key issue in Algorithm 9 is that the length of D_i can be 2^i, making the merge algorithm take exponential time. To fix this, we employ a $Trim$ function to shorten the list length. The idea is that if some combinations of features have similar effect on $q(x)$, only one combination is considered. This means that with a trimming parameter δ, for any element d_i removed from D_i, there is an element d_j that approximates d_i, that is, $\frac{Retrieve(d_i)}{1+\delta} \leq Retrieve(d_j) \leq Retrieve(d_i)$. Notice that the $Trim$ action can only be done for features that have no common bases to avoid missing qualified feature combination. Therefore, *GenFeaGroup* algorithm is applied to group the features that need to be added as a whole before $Trim$. *AddFea* algorithm then helps to form different feature combinations and guarantees that at most k features are considered.

In addition to the approximation algorithm above, we consider two others: an optimal *branch-and-bound* search with worst-case exponential running time, and a greedy heuristic (*Greedy*). In the *branch-and-bound* scheme, we search in the space of feature changes to x. At any node with height l, we have thereby changed l features in x, and the utility of the attacker in this subtree is therefore bounded above by $e^{-\delta l}$ (since $1 - q(x') \leq 1$). This bound is used in pruning much of the search tree once a good solution using relatively few modifications is found. In the greedy heuristic, we start with x and iteratively flip features one at a time, flipping a feature that yields the greatest decrease in $q(x')$ each time.

We used TREC spam corpora to experimentally compare the three approaches to com-

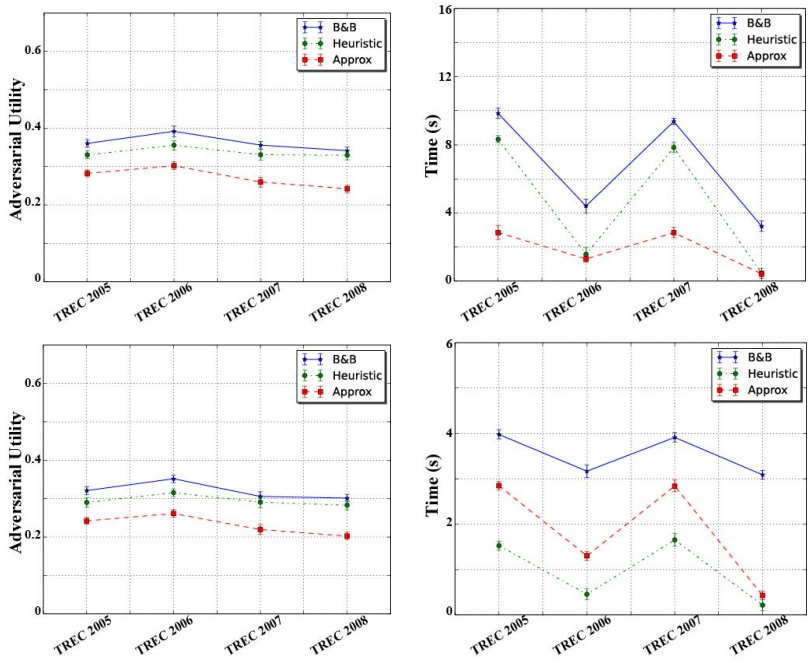


Figure 7.3: Comparision of expected adversary utility (left) and algorithm runtime (right), for the three adversrial evasion algorithms. Top: $\delta = 1$, $\varepsilon = 0.01$. Bottom: $\delta = 3$, $\varepsilon = 0.01$

puting adversarial evasion: the ApproxEvasion algorithm,[1] branch-and-bound, and greedy heuristic. The results, shown in Figure 8.3, suggest (somewhat surprisingly) that the simple greedy heuristic offers a good balance between running time and quality: it is faster, usually quite significantly, than branch-and-bound, and loses less in solution quality than the approximation algorithm. Consequently, our implementation of AAS features the evasion oracle $\mathscr{O}$ which runs the greedy heuristic.

7.4 Experiments

To evaluate the efficacy of the proposed AAS algorithm for approximating optimal randomized operational decisions in adversarial classification settings, we compare the optimized utility of defender with the state of the art. The results below use 100 features, with additional results (using 500 features over the same domain) presented in the appendix.

[1]Whie ApproxEvasion cannot be used directly, it can be adapted using a linear search in the space of thresholds k along with the same bound as used in branch-and-bound to truncate the search.

In the experiments, we use the TREC spam corpora from 2005-2008.[2] First, we evaluate the performance of AAS as a spam filtering task to compare the classification accuracy with the state of the art alternatives [122, 84]. In this first set of experiments, which are used to test robustness to *naturally observed* spam evolution, we apply a fold of TREC 2005 data as training and evaluate and test on the test fold for the TREC 2005 and 2006-2008 corpora (in other words, we train on "current" data, and observe performance on "future" data). We compare our approach against using a static classifier it is based upon, where the pair of the form $\{C, AAS(C)\}$, consists of a static classifier C which is used to learn $p(x)$ for our model, and $AAS(C)$ corresponding to our AAS algorithm that utilizes C. Here we use the normalized utility as $U_D = 1 - \frac{w|X^+|+|X^-|}{w|X_{TN}|+|X_{TP}|}$, where $|X_{TN}|$ is the number of true negatives, while $|X_{TP}|$ the number of true positives. $|X^-| = \sum_x y(x)(1-q(x))$ represents the expected number of false negatives, while $|X^+| = \sum_x (1-y(x))q(x)$ the expected number of false positives; $w = \frac{G}{V}$.

Figure 8.4 shows that when the budget constraint is tight, our approach significantly outperforms the baselines. From Figure 8.5 it can also be observed how the cost of adversary matters. When we fix $G(x) = 1$ and vary $V(x) = V$ (keeping it constant for all x), our approach still consistently outperforms alternatives.

In the next set of experiments, we simulated a counterfactual of sophisticated evasion attacks, deployed according to our model, drawing the same comparisons as above, but now treating each year in the TREC data as distinct (in other words, we consider each year as "current", and then simulate an evasion attack independently for each year). From Figure 8.6 and 8.7 we can see that our proposed method significantly outperforms the alternatives on different datasets across both alternative budget constraints and value models.

It is, of course, not surprising that our proposed approach outperforms alternative methods *in terms of the objective it tries to optimize*. A natural question, however, is whether this approach is robust to errors which would be inevitable in its practical deployment. To

[2] Our choice of TREC corpora for this evaluation is due primarily to its longitudinal nature, which is key for a subset of our experiments.

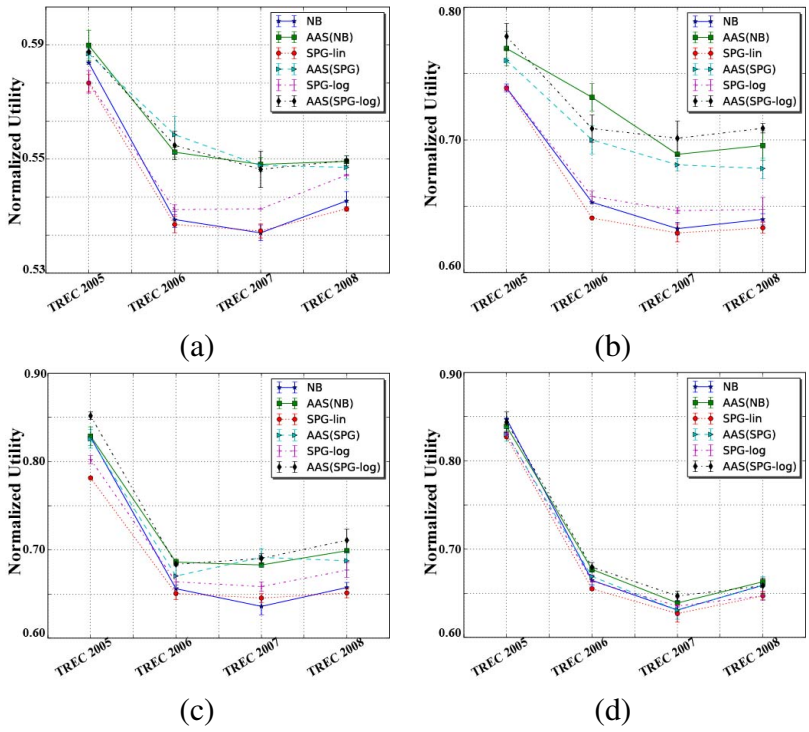


Figure 7.4: Comparison of normalized utility on TREC data, trained on year 2005, and tested on years 2005-2008. Our method is labeled as $AAS(\cdot)$, where the parameter is the classifier that serves to provide $p(x)$. The following parameters are used: $\delta = 1$, $V(x) = G(x) = 1$, $P_A = 1$ (a) c=0.1; (b) c=0.3; (c) c=0.5; (d) c=0.9.

evaluate the robustness of our algorithm, we introduce errors into our attacker model. First, we introduce an error $\varphi = 0.2$ into the attacker model, so that $\hat{\delta} = \delta + \varphi$, where $\hat{\delta}$ is the "observed" and δ the actual model parameter. Figure 8.8 and 8.9 show that our approach still outperforms the state of the art alternatives even when harmed by very substantial inaccuracy in the model parameter estimates.

Next, we consider robustness to a *qualitative* rather than *quantitative* error in adversarial model. To simulate this, we solve our model as before, but evaluate the solutions $q(x)$ by assuming an adversary's utility model actually decays polynomially as

$$Q_{poly}(x,x') = \frac{1}{1+\delta\|x-x'\|}.$$

The results, shown in Figure 8.10, demonstrate that our model is robust even when the

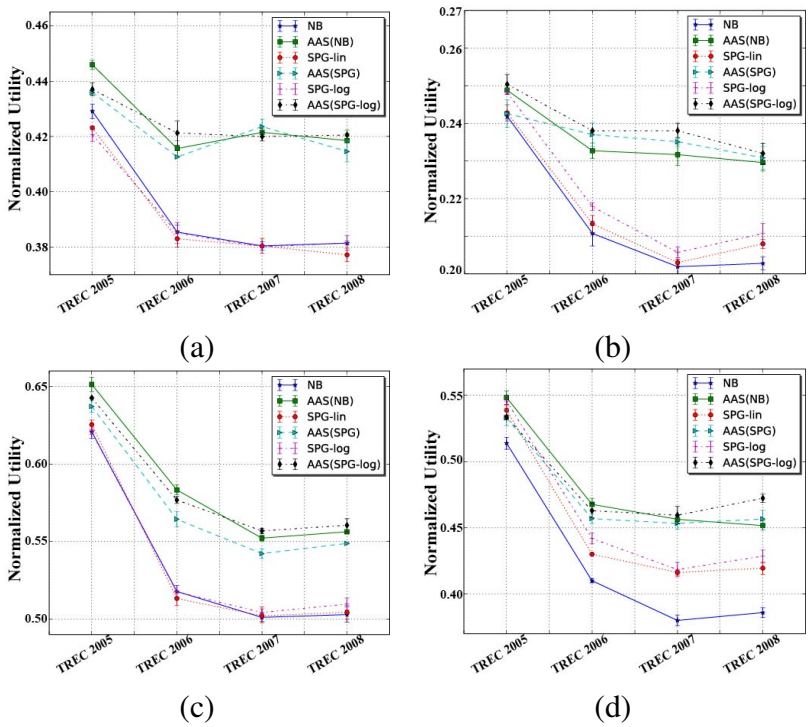


Figure 7.5: Comparison of normalized utility on TREC data, trained on year 2005, and tested on years 2005-2008. Our method is labeled as $AAS(\cdot)$, where the parameter is the classifier that serves to provide $p(x)$. The following parameters are used: $\delta = 1$, $G(x) = 1$, $P_A = 1$ (a) $V(x) = 2$, c=0.1; (b) $V(x) = 10$, c=0.1; (c) $V(x) = 2$, c=0.3; (d) $V(x) = 10$, c=0.3.

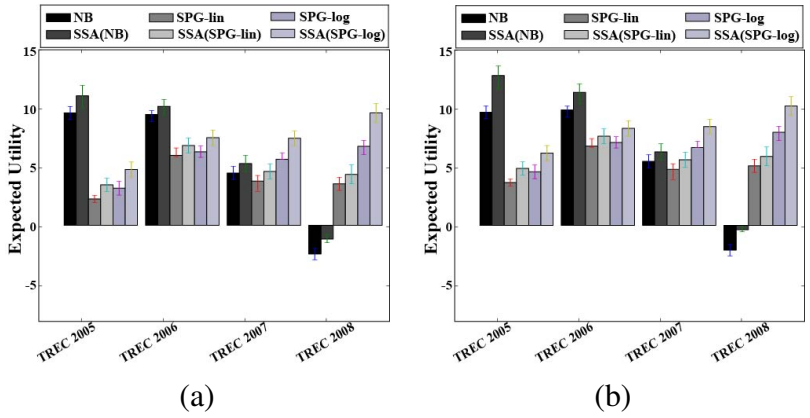
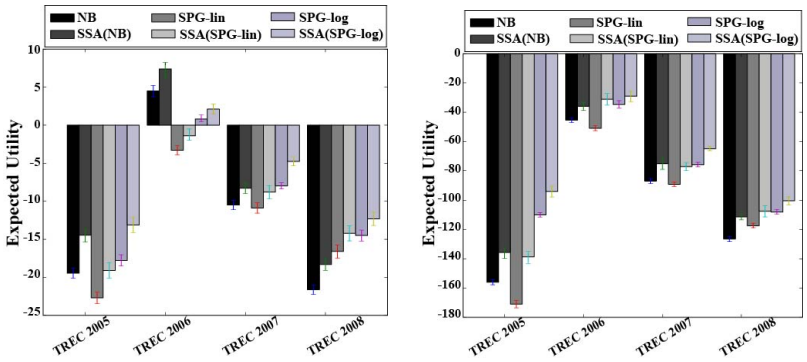


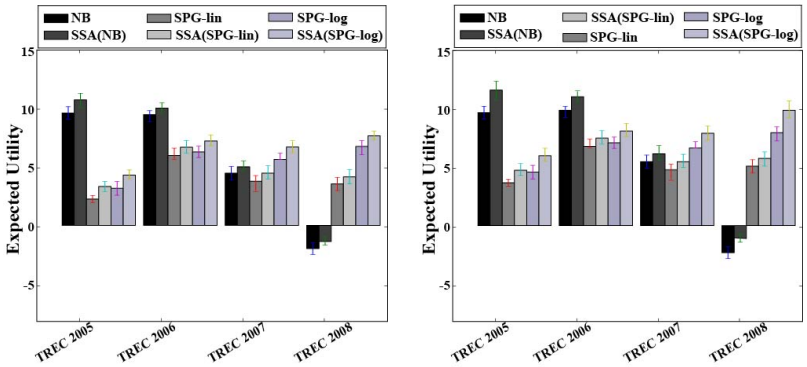
Figure 7.6: Comparison of the expected utility assuming $P_A = 1$, $V(x) = G(x) = 1$; (a) $c = 0.1$; (b) $c = 0.3$.

assumption about the adversary utility model is fundamentally incorrect.



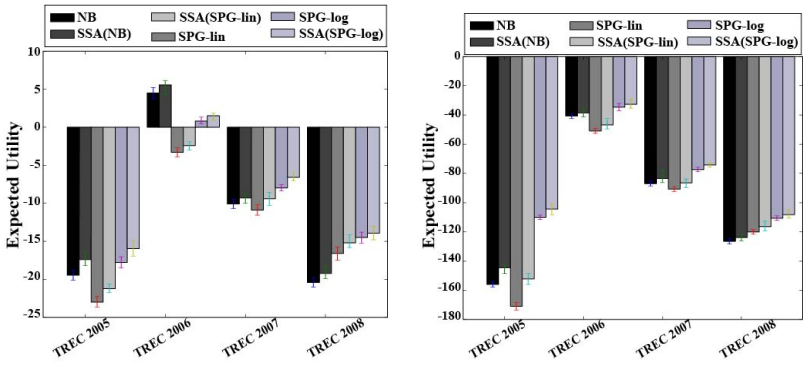
(a) (b)

Figure 7.7: Comparison of the expected utility assuming $P_A = 1$; (a) $V(x) = 2$; (b) $V(x) = 10$. $c = 0.3$.



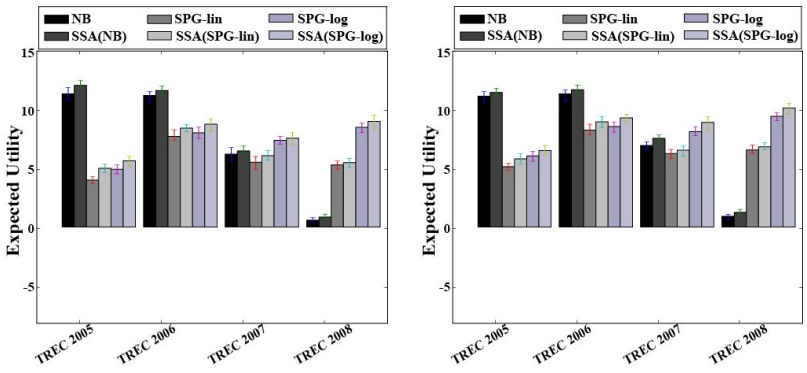
(a) (b)

Figure 7.8: Comparison of the expected utility assuming $P_A = 1$, introducing parameter error with 0.2 for δ; (a) $c = 0.1$; (b) $c = 0.3$.



(a) (b)

Figure 7.9: Comparison of the expected utility assuming $P_A = 1$, introducing parameter error with 0.2 for δ; (a) $V(x) = 2$; (b) $V(x) = 10$. $c = 0.3$.



(a) (b)

Figure 7.10: Comparison of the expected utility assuming $P_A = 1$, introducting adversarial model error; (a) $c = 0.1$; (b) $c = 0.3$.

7.5 Summary of Contributions

This work presented a general approach for computing optimal randomized decisions in adversarial classification settings. We solve the resulting intractably large problem by applying a finite set of basis functions and using constraint generation which leverages high-quality approximation of optimal adversarial classifier evasion. The proposed method outperforms than the state of the art alternatives on several metrics, is robust to errors (including qualitative mistakes in modeling assumptions) and its advantages are more apparent when operational decisions are costly. Moreover, by conceptually separating the problem of prediction (of adversary's preferences) and optimal operational decisions, the approach can both make use of off-the-shelf machine learning techniques, and naturally embed randomization. While the use of machine learning in adversarial settings, such as network intrusion detection, is still quite limited, our approach may pave the way for bridging the gap between algorithmic advances and operational deployment of such systems.

In all, the contributions are as listed following: 1) a general framework for optimizing operational decisions based on machine learning predictions; 2) a linear programming formulation to compute optimal randomized operational decisions under budget constraints; 3) an approach for scalable parity-basis approximation of operational decision function; 4)

a model of attacker evasion and methods approaches for approximating optimal evasion; and 5) an extensive evaluation of our approach, which we show to significantly outperform the state of the art.

Part II Data manipulation analysis

Chapter 8

DATA POISONING ATTACKS FOR FACTORIZATION BASED COLLABORATIVE FILETERING

In addition to the evasion attacks we studied in the previous chapters, in this chapter we start to consider another kind of attack, poisoning attacks. By poisoning the training data, the adversaries can mislead the trained learning systems and cause harmful results. Therefore here we study how the poisoning attack compromises the learner for a particular application, recommendation system. Recommendation and collaborative filtering systems are important in nowadays information and e-commerce applications. As these systems are becoming increasingly popular in industry, their outputs could affect business decision making and hence there is always incentive for an adversarial party to compromise the availability or integrity of such systems. Therefore here we consider a data poisoning attacking model where an attacker is capable of injecting "malicious" data into a recommendation system. We discuss how can an attacker generate malicious data so as to maximize his/her malicious objectives, while at the same time mimic normal users' behaviors to avoid being detected. Efficient solutions are presented for two popular factorization based collaborative filtering algorithms: the *alternative minimization* formulation and the *nuclear norm minimization* method. Finally, we test the effectiveness of our proposed algorithms on real-world dataset and discuss potential defensive strategies.

8.1 Overview

Recommendation systems are prevalent in the era of world wide web and big data. Some typical applications include recommendation systems for movies, books, restaurants, and hotels. At a higher level, making recommendations involves automatically fitering each

user's preferences of items based on the user as well as other users' known preferences of a small portion of items in the database. In machine learning such problems are usually referred to as *collaborative filtering* or *matrix completion*, where the known users' preference are abstracted into an incomplete user-by-item matrix, and the goal is to complete the matrix and subsequently make new item recommendations for each user. Existing approaches in the literature include nearest-neighbor methods, where a user's (item's) preference is determined by other users (items) with similar profiles [177], and factorization-based methods where the incomplete preference matrix is assumed to be approximately low-rank [178, 179].

As recommendation systems play an ever increasing role in current information and e-commerce systems, they are susceptible to an enormous risk of being maliciously attacked. One particular form of attacks is called *data poisoning*, in which a malicious party creates dummy (malicious) users in a recommendation system with carefully chosen item preferences (i.e., data) such that the effectiveness or credibility of the system is maximally tampered with. For example, an attacker might attempt to make recommendations that are as different as possible from those that would otherwise be made by the recommendation system. In another, more subtle, example, the attacker is associated with the producer of a specific movie or product, who may wish to increase or decrease the popularity of a certain item. In both cases, the credibility of a recommendation system is harmed by the malicious activities, which could lead to significant ecomonic loss as well. Due to the open nature of recommendation systems and their reliance on user-specified judgments for building profiles, various forms of attacks are possible and have been discussed, such as the random attack and random product push/nuke attack [180, 181]. However, these attacks are not formally analyzed and cannot be optimized according to specific collaborative filtering algorithms. As it is not difficult for attackers to determine the defender's filtering algorithm or even its parameters settings, this can lead one to under-estimate the attacker's ability and result in substantial loss.

Here we provide a systematic approach to compute near-optimal data poisoning attacks for factorization-based collaborative filtering/recommendation models. We focus on two most popular algorithms: *alternating minimization* [182] and *nuclear norm minimization* [179]. Our main contributions are as follows:

Comprehensive characterization of attacker utilities: We characterize several attacker utilities, which include *availability attacks*, where prediction error is increased, and *integrity attacks*, where item-specific objectives are considered. Optimal attack strategies for all utilities can be computed under a unified optimization framework.

Novel gradient computations: Building upon existing gradient-based data poisoning frameworks [183, 184, 185], we develop novel methods for gradient computation based on first-order KKT conditions for two widely used algorithms: alternating minimization [182] and nuclear norm minimization [178]. The resulting derivations are highly non-trivial; in addition, to our knowledge this work is the first to give systematic data poisoning attacks for problems involving non-smooth nuclear norm type objectives.

Mimicking normal user behaviors: For data poisoning attacks, most prior work focuses on maximizing attacker's utility. A less investigated problem is how to synthesize malicious data points that are hard for a defender to detect. This work provides a novel technique based on *stochastic gradient Langevin dynamics* optimization [186] to produce malicious users that mimic normal user behaviors in order to avoid detection. The illustration of such poisoning attack is shown in Figure 9.1.

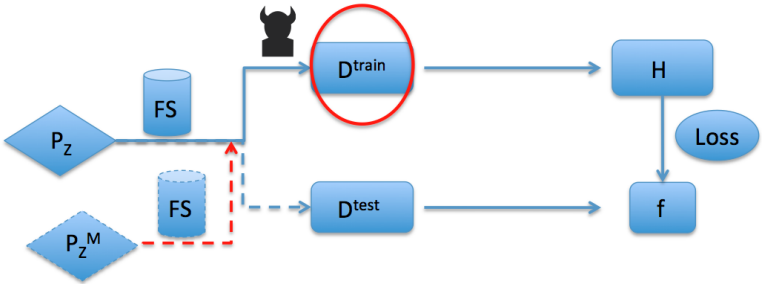


Figure 8.1: Illustration for poisoning attack within learning systems

8.2 Related Work

There has been extensive prior research concerning the security of machine learning algorithms [44, 45, 43, 155]. Much of this line of work is dedicated to methods that deal with evasion attacks, where the attacker can only manipulate the testing data. Few research is done along the direction of data poisoning attacks, where the attacker is only allowed to manipulate or inject malicious data in order to break the targeted learning systems. Such poisoning attacks may lead to great loss, especially in health-care and biomedicine domains where the consequences of poisoning attacks could be life-threatening and may cause distrust [187].

Biggio et al. pioneered the research of optimizing malicious data-driven attacks for kernel-based learning algorithms such as SVM [188]. The key optimization technique is to approximately compute implicit gradients of the solution of an optimization problem based on first-order KKT conditions. Similar techniques were later generalized to optimize data poisoning attacks for several other important learning algorithms, such as Lasso regression [183], topic modeling [184] and autoregressive models [189]. The reader may refer to [185] for a general algorithmic framework of the abovementioned methods.

In terms of collaborative filtering/matrix completion, there is another line of established research that focuses on *robust matrix completion*, in which a small portion of elements or rows in the underlying low-rank matrix is assumed to be arbitrarily perturbed [190, 191, 192, 193]. Typically an *exact* low-rank "signal matrix" is assumed for recovery purposes, which never holds true in practice. In addition, robust matrix completion is usually accopmlished by solving a mixed-norm convex optimization problem. Such procedures require significant amounts of computation and are hard to scale to large data sets. Also, it is unclear what types of outliers perturb the matrix completion solution the most and how much such perturbation is possible without additional defending strategies. One exception is [194] in which the stability of alternating minimization solutions was analyzed with respect to malicious data manipulations. However, [194] assumes the global optimal

solution of alternating minimization can ba obtained, which is rarely true in practice. Finally, [195] analyzed adversarial learning of *item-based* collaborative filtering systems, which fundementally differ from decomposition based methods considered in this work.

8.3 Preliminaries

We first set up the collaborative filtering/matrix completion problem and give an overview of existing low-rank factorization based approaches. Let $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ be a data matrix consisting of m rows and n columns. Each row represents a user and each column represents an item, for example, movies in Netflix or commodities for Amazon. $\boldsymbol{M}_{ij}$ for $i \in [m]$ and $j \in [n]$ would then correspond to the rating the ith user gives for the jth item. Typically, only a very small portion of $\boldsymbol{M}$ is observed, as each user only rates few items in the database. We use $\Omega = \{(i,j) : \boldsymbol{M}_{ij} \text{ is observed}\}$ to denote all observable entries in $\boldsymbol{M}$ and assume that $|\Omega| \ll mn$. We also use $\Omega_i \subseteq [n]$ and $\Omega'_j \subseteq [m]$ for columns (rows) that are observable at the ith row (jth column). The goal of collaborative filtering (also referred to as *matrix completion* in the statistical learning literature [178]) is then to recover the complete matrix $\boldsymbol{M}$ from few observations $\boldsymbol{M}_\Omega$.

The matrix completion problem is in general ill-posed as it is impossible to complete an arbitrary matrix with partial observations. As a result, additional assumptions are imposed on the underlying data matrix $\boldsymbol{M}$. One standard assumption is that $\boldsymbol{M}$ is very close to an $m \times n$ rank-k matrix with $k \ll \min(m,n)$. Under such assumptions, the complete matrix $\boldsymbol{M}$ can be recovered by solving the following optimization problem:

$$\min_{\boldsymbol{X}\in\mathbb{R}^{m\times n}} \|\mathscr{R}_\Omega(\boldsymbol{M}-\boldsymbol{X})\|_F^2, \quad s.t.\ \operatorname{rank}(\boldsymbol{X}) \le k, \tag{8.1}$$

where $\|\boldsymbol{A}\|_F^2 = \sum_{i,j} \boldsymbol{A}_{ij}^2$ denotes the squared Frobenious norm of matrix $\boldsymbol{A}$ and $[\mathscr{R}_\Omega(\boldsymbol{A})]_{ij}$ equals $\boldsymbol{A}_{ij}$ if $(i,j) \in \Omega$ and 0 otherwise. Unfortunately, the feasible set in Eq. (9.1) is non-convex, making the optimimzation problem difficult to solve. There has been an extensive

prior literature on approximately solving Eq. (9.1) and/or its surrogates that lead to two standard approaches: alternating minimization and nuclear norm minimization. For the first approach, one considers the following problem:

$$\min_{\boldsymbol{U}\in\mathbb{R}^{m\times k},\boldsymbol{V}\in\mathbb{R}^{n\times k}} \left\{ \|\mathscr{R}_{\Omega}(\boldsymbol{M}-\boldsymbol{U}\boldsymbol{V}^{\top})\|_F^2 + 2\lambda_U\|\boldsymbol{U}\|_F^2 + 2\lambda_V\|\boldsymbol{V}\|_F^2 \right\}. \quad (8.2)$$

Eq. (9.2) is equivalent to Eq. (9.1) when $\lambda_U = \lambda_V = 0$. In practice people usually set both regularization parameters λ_U and λ_V to be small positive constants in order to avoid large entries in the completed matrix and also improve convergence. Since Eq. (9.2) is bi-convex in $\boldsymbol{U}$ and $\boldsymbol{V}$, an *alternating minimization* procedure can be applied. Alternatively, one solves a *nuclear-norm minimization* problem

$$\min_{\boldsymbol{X}\in\mathbb{R}^{m\times n}} \|\mathscr{R}_{\Omega}(\boldsymbol{M}-\boldsymbol{X})\|_F^2 + 2\lambda\|\boldsymbol{X}\|_*, \quad (8.3)$$

where $\lambda > 0$ is a regularization parameter and $\|\boldsymbol{X}\|_* = \sum_{i=1}^{\text{rank}(\boldsymbol{X})} |\sigma_i(\boldsymbol{X})|$ is the nuclear norm of $\boldsymbol{X}$, which acts as a convex surrogate of the rank function. Eq. (9.3) is a convex optimization function and can be solved using an iterative singular value thresholding algorithm [179]. It can be shown that both methods in Eq. (9.2) and (9.3) provably approximate the true underlying data matrix $\boldsymbol{M}$ under certain conditions [182, 178].

8.4 The Attack Model

In this section we describe the data poisoning attacking model considered in this study. We assume that the attacker has access to all observable matrix entries $\boldsymbol{M}_{\Omega}$ and has full knowledge of the collaborative filtering algorithm used as well as the way all algorithm parameters (e.g., the regularization parameter λ in Eq. (9.3)) are set. For a data matrix consisting of m users and n items, the attacker is capable of adding αm malicious users to the training data matrix, and each malicious user is allowed to report his/her preference on

at most B items with each preference bounded in the range $[-\Lambda, \Lambda]$. Here $\alpha \ll 1$, $B \ll n$ and $\Lambda < \infty$ are budget parameters in the attacking model.

Before proceeding to describe the attacker's goals, we first define a few notations for the ease of presentation. We use $\boldsymbol{M} \in \mathbb{R}^{m \times n}$ to denote the original data matrix and $\widetilde{\boldsymbol{M}} \in \mathbb{R}^{m' \times n}$ to denote the data matrix of all $m' = \alpha m$ malicious users. Let $\widetilde{\Omega}$ be the set of non-zero entries in $\widetilde{\boldsymbol{M}}$ and $\widetilde{\Omega}_i \subseteq [n]$ be all items that the ith malicious user rated. According to our attack models, $|\widetilde{\Omega}_i| \leq B$ for every $i \in \{1, \cdots, m'\}$ and $\|\widetilde{\boldsymbol{M}}\|_{\max} = \max |\widetilde{\boldsymbol{M}}_{ij}| \leq \Lambda$.

Let $\boldsymbol{\Theta}_\lambda(\widetilde{\boldsymbol{M}}; \boldsymbol{M})$ be the optimal solution computed jointly on the original and poisoned data matrices $(\widetilde{\boldsymbol{M}}; \boldsymbol{M})$ using regularization parameters λ. For example, Eq. (9.2) becomes

$$\boldsymbol{\Theta}_\lambda(\widetilde{\boldsymbol{M}}; \boldsymbol{M}) = \underset{\boldsymbol{U}, \widetilde{\boldsymbol{U}}, \boldsymbol{V}}{\arg\min} \|\mathscr{R}_\Omega(\boldsymbol{M} - \boldsymbol{U}\boldsymbol{V}^\top)\|_F^2 + \|\mathscr{R}_{\tilde{\Omega}}(\widetilde{\boldsymbol{M}} - \widetilde{\boldsymbol{U}}\boldsymbol{V}^\top)\|_F^2 + 2\lambda_U(\|\boldsymbol{U}\|_F^2 + \|\widetilde{\boldsymbol{U}}\|_F^2) + 2\lambda_V\|\boldsymbol{V}\|_F^2 \quad (8.4)$$

where the resuling $\boldsymbol{\Theta}$ consists of low-rank latent factors $\boldsymbol{U}, \widetilde{\boldsymbol{U}}$ for normal and malicious users as well as $\boldsymbol{V}$ for items. Simiarly, for the nuclear norm minimization formulation in Eq. (9.3), we have

$$\boldsymbol{\Theta}_\lambda(\widetilde{\boldsymbol{M}}; \boldsymbol{M}) = \underset{\boldsymbol{X}, \widetilde{\boldsymbol{X}}}{\arg\min} \|\mathscr{R}_\Omega(\boldsymbol{M} - \boldsymbol{X})\|_F^2 + \|\mathscr{R}_{\tilde{\Omega}}(\widetilde{\boldsymbol{M}} - \widetilde{\boldsymbol{X}})\|_F^2 + 2\lambda\|(\boldsymbol{X}; \widetilde{\boldsymbol{X}})\|_*, \quad (8.5)$$

where $\boldsymbol{\Theta} = (\boldsymbol{X}, \widetilde{\boldsymbol{X}})$.

Let $\widehat{\boldsymbol{M}}(\boldsymbol{\Theta})$ be the matrix estimated from learnt model $\boldsymbol{\Theta}$. For example, for Eq. (9.4) we have $\widehat{\boldsymbol{M}}(\boldsymbol{\Theta}) = \boldsymbol{U}\boldsymbol{V}^\top$ and for Eq. (9.5) we have $\widehat{\boldsymbol{M}}(\boldsymbol{\Theta}) = \boldsymbol{X}$. The goal of the attacker is then to find optimal malicious users $\widetilde{\boldsymbol{M}}^*$ such that

$$\widetilde{\boldsymbol{M}}^* \in \operatorname{argmax}_{\widetilde{\boldsymbol{M}} \in \mathbb{M}} R(\widehat{\boldsymbol{M}}(\boldsymbol{\Theta}_\lambda(\widetilde{\boldsymbol{M}}; \boldsymbol{M})), \boldsymbol{M}). \quad (8.6)$$

Here $\mathbb{M} = \{\widetilde{\boldsymbol{M}} \in \mathbb{R}^{m' \times n} : |\tilde{\Omega}_i| \leq B, \|\widetilde{\boldsymbol{M}}\|_{\max} \leq \Lambda\}$ is the set of all feasible poisoning attacks

discussed earlier in this section and $R(\widehat{\boldsymbol{M}}, \boldsymbol{M})$ denotes the attacker's utility for diverting the collaborative filtering algorithm to predict $\widehat{\boldsymbol{M}}$ on an original data set $\boldsymbol{M}$, with the help of few malicious users $\widetilde{\boldsymbol{M}}$. Below we list several typical attacker utilities: **Availability attack** the attacker wants to maximize the error of the collaborative filtering system, and eventually render the system useless. Suppose $\overline{\boldsymbol{M}}$ is the prediction of the collaborative filtering system without data poisoning attacks. [1] The utility function is then defined as the total amount of perturbation of predictions between $\overline{\boldsymbol{M}}$ and $\widehat{\boldsymbol{M}}$ (predictions after poisoning attacks) on unseen entries Ω^C:

$$R^{\text{av}}(\widehat{\boldsymbol{M}}, \boldsymbol{M}) = \|\mathscr{R}_{\Omega^C}(\widehat{\boldsymbol{M}} - \overline{\boldsymbol{M}})\|_F^2. \tag{8.7}$$

Integrity attack in this model the attacker wants to boost (or reduce) the popularity of a (subset) of items. Suppose $J_0 \subseteq [n]$ is the subset of items the attacker is interested in and $w : J_0 \to \mathbb{R}$ is a pre-specified weight vector by the attacker. The utility function is

$$R^{\text{in}}_{J_0,w}(\widehat{\boldsymbol{M}}, \boldsymbol{M}) = \sum_{i=1}^{m} \sum_{j \in J_0} w(j) \widehat{\boldsymbol{M}}_{ij}. \tag{8.8}$$

Hybrid attack a hybrid loss function can also be defined:

$$R^{\text{hybrid}}_{J_0,w,\mu}(\widehat{\boldsymbol{M}}, \boldsymbol{M}) = \mu_1 R^{\text{av}}_{J_0,w}(\widehat{\boldsymbol{M}}, \boldsymbol{M}) + \mu_2 R^{\text{in}}(\widehat{\boldsymbol{M}}, \boldsymbol{M}), \tag{8.9}$$

where $\mu = (\mu_1, \mu_2)$ are coefficients that trade off the availability and integrity attack objectives. In addition, μ_1 could be negative, which models the case when the attacker wants to leave a "light trace": the attacker wants to make his item more popular while making the other recommendations of the system less perturbed to avoid detection.

[1] Note that when the collaborative filtering algorithm and its parameters are set, $\overline{\boldsymbol{M}}$ is a function of observed entries $\mathscr{R}_{\Omega}(\boldsymbol{M})$.

8.5 Computing Optimal Attack Strategies

We describe practical algorithms to solve the optimization problem in Eq. (9.6) for optimal attack strategy $\widetilde{\boldsymbol{M}}^*$ that maximizes attacker's utility. We first consider the alternating minimization formulation in Eq. (9.4) and derive a projective gradient ascent method that solves for the corresponding optimal attack strategy. Similar derivations are then extended to the nuclear norm minimization formulation in Eq. (9.5). Finally, we discuss how to design malicious users that mimic normal users' behaviors in order to avoid potential detection from the defender side.

8.5.1 Attacking Alternating Minimization

We use the *projective gradient ascent* (PGA) method for solving the optimization problem in Eq. (9.6) with respect to the alternating minimization formulation in Eq. (9.4): in iteration t we update $\widetilde{\boldsymbol{M}}^{(t)}$ as follows:

$$\widetilde{\boldsymbol{M}}^{(t+1)} = \mathrm{Proj}_{\mathbb{M}}\left(\widetilde{\boldsymbol{M}}^{(t)} + s_t \cdot \nabla_{\widetilde{\boldsymbol{M}}} R(\widehat{\boldsymbol{M}}, \boldsymbol{M})\right), \tag{8.10}$$

where $\mathrm{Proj}_{\mathbb{M}}(\cdot)$ is the projection operator onto the feasible region $\mathbb{M}$ and s_t is the step size in iteration t. Note that the estimated matrix $\widehat{\boldsymbol{M}}$ depends on the model $\boldsymbol{\Theta}_\lambda(\widetilde{\boldsymbol{M}}; \boldsymbol{M})$ learnt on the joint data matrix, which further depends on the malicious users $\widetilde{\boldsymbol{M}}$. Since the constraint set $\mathbb{M}$ is highly non-convex, we generate B items uniformly at random for each malicious user to rate. The $\mathrm{Proj}_{\mathbb{M}}(\cdot)$ operator then reduces to projecting each malicious users' rating vector onto an ℓ_∞ ball of diameter Λ, which can be easily evaluated by truncating all entries in $\widetilde{\boldsymbol{M}}$ at the level of $\pm\Lambda$.

We next show how to (approximately) compute $\nabla_{\widetilde{\boldsymbol{M}}} R(\widehat{\boldsymbol{M}}, \boldsymbol{M})$. This is challenging because one of the arguments in the loss function involves an implicit optimization problem.

We first apply chain rule to arrive at

$$\nabla_{\widetilde{M}} R(\widehat{M}, M) = \nabla_{\widetilde{M}} \Theta_\lambda(\widetilde{M}; M) \nabla_{\Theta} R(\widehat{M}, M). \tag{8.11}$$

The second gradient (with respect to Θ) is easy to evaluate, as all loss functions mentioned in the previous section are smooth and differentiable. Detailed derivation of $\nabla_{\Theta} R(\widehat{M}, M)$ is deferred to Appendix. On the other hand, the first gradient term term is much harder to evaluate because $\Theta_\lambda(\cdot)$ is an optimization procedure. Inspired by [183, 184, 185], we exploit the KKT conditions of the optimization problem $\Theta_\lambda(\cdot)$ to approximately compute $\nabla_{\widetilde{M}} \Theta_\lambda(\widetilde{M}; M)$. More specifically, the optimal solution $\Theta = (U, \widetilde{U}, V)$ of Eq. (9.4) satisfy

$$\begin{aligned}
\lambda_U u_i &= \sum_{j \in \Omega_i} (M_{ij} - u_i^\top v_j) v_j; \\
\lambda_U \tilde{u}_i &= \sum_{j \in \widetilde{\Omega}_i} (\widetilde{M}_{ij} - \tilde{u}_i^\top v_j) v_j; \\
\lambda_V v_j &= \sum_{i \in \Omega'_j} (M_{ij} - u_i^\top v_j) u_i + \sum_{i \in \widetilde{\Omega}'_j} (\widetilde{M}_{ij} - \tilde{u}_i^\top v_j) \tilde{u}_i,
\end{aligned}$$

where $u_i, \tilde{u}_i$ are the ith rows (of dimension k) in U or $\widetilde{U}$ and v_j is the jth row (also of dimension k) in V. Subsequently, $\{u_i, \tilde{u}_i, v_j\}$ can be expressed as functions of the original and malicious data matrices M and $\widetilde{M}$. Using the fact that $(a^\top x)a = (aa^\top)x$ and M does not change with $\widetilde{M}$, we obtain

$$\frac{\partial u_i(\widetilde{M})}{\partial \widetilde{M}_{ij}} = 0; \quad \frac{\partial \tilde{u}_i(\widetilde{M})}{\partial \widetilde{M}_{ij}} = \left(\lambda_U I_k + \Sigma_U^{(i)}\right)^{-1} v_j;$$

$$\frac{\partial v_j(\widetilde{M})}{\partial \widetilde{M}_{ij}} = \left(\lambda_V I_k + \Sigma_V^{(j)}\right)^{-1} u_i.$$

Here $\Sigma_U^{(i)}$ and $\Sigma_V^{(j)}$ are defined as

Algorithm 14 Optimizing $\widetilde{\boldsymbol{M}}$ via PGA

1: **Input**: Original partially observed $m \times n$ data matrix $\boldsymbol{M}$, algorithm regularization parameter λ, attack budget parameters α, B and Λ, attacker's utility function R, step size $\{s_t\}_{t=1}^{\infty}$.
2: **Initialization**: random $\widetilde{\boldsymbol{M}}^{(0)} \in \mathbb{M}$ with both ratings and rated items uniformly sampled at random; $t = 0$.
3: **while** $\widetilde{\boldsymbol{M}}^{(t)}$ does not converge **do**
4: Compute the optimal solution $\boldsymbol{\Theta}_\lambda(\widetilde{\boldsymbol{M}}^{(t)}; \boldsymbol{M})$.
5: Compute gradient $\nabla_{\widetilde{\boldsymbol{M}}} R(\widehat{\boldsymbol{M}}, \boldsymbol{M})$ using Eq. (9.10).
6: Update: $\widetilde{\boldsymbol{M}}^{(t+1)} = \text{Proj}_{\mathbb{M}}(\widetilde{\boldsymbol{M}}^{(t)} + s_t \nabla_{\widetilde{\boldsymbol{M}}} R)$.
7: $t \leftarrow t + 1$.
8: **end while**
9: **Output**: $m' \times n$ malicious matrix $\widetilde{\boldsymbol{M}}^{(t)}$.

$$\boldsymbol{\Sigma}_U^{(i)} = \sum_{j \in \Omega_i \cup \widetilde{\Omega}_i} \boldsymbol{v}_j \boldsymbol{v}_j^\top, \quad \boldsymbol{\Sigma}_V^{(j)} = \sum_{i \in \Omega'_j \cup \widetilde{\Omega}'_j} \boldsymbol{u}_i \boldsymbol{u}_i^\top. \tag{8.12}$$

A framework of the proposed optimization algorithm is described in Algorithm 14.

8.5.2 Attacking Nuclear Norm Minimization

We extend the projective gradient ascent algorithm in Sec. 9.5.1 to compute optimal attack strategies for the nuclear norm minimization formulation in Eq. (9.5). Since the objective in Eq. (9.5) is convex, the global optimal solution $\boldsymbol{\Theta} = (\boldsymbol{X}, \widetilde{\boldsymbol{X}})$ can be obtained by conventional convex optimization procedures such as proximal gradient descent (a.k.a. singular value thresholding [179] for nuclear norm minimization). In addition, the resulting estimation $(\boldsymbol{X}; \widetilde{\boldsymbol{X}})$ is low rank due to the nuclear norm penalty [178]. Suppose $(\boldsymbol{X}; \widetilde{\boldsymbol{X}})$ has rank $\rho \leq \min(m, n)$. We use $\boldsymbol{\Theta}' = (\boldsymbol{U}, \widetilde{\boldsymbol{U}}, \boldsymbol{V}, \boldsymbol{\Sigma})$ as an alternative characterization of the learnt model with reduced number of parameters. Here $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ and $\widetilde{\boldsymbol{X}} = \widetilde{\boldsymbol{U}}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ are singular value decompositions of $\boldsymbol{X}$ and $\widetilde{\boldsymbol{X}}$; that is, $\boldsymbol{U} \in \mathbb{R}^{m \times \rho}$, $\widetilde{\boldsymbol{U}} \in \mathbb{R}^{m' \times \rho}$, $\boldsymbol{V} \in \mathbb{R}^{n \times \rho}$ have orthornormal columns and $\boldsymbol{\Sigma} = \boldsymbol{diag}(\sigma_1, \cdots, \sigma_\rho)$ is a non-negative diagonal matrix.

To compute the gradient $\nabla_{\widetilde{\boldsymbol{M}}} R(\widehat{\boldsymbol{M}}, \boldsymbol{M})$, we again apply the chain rule to decompose

the gradient into two parts:

$$\nabla_{\widetilde{M}} R(\widehat{M}, M) = \nabla_{\widetilde{M}} \Theta'_\lambda(\widetilde{M}; M) \nabla_{\Theta'} R(\widehat{M}, M). \tag{8.13}$$

Similar to Eq. (9.11), the second gradient term $\nabla_{\Theta'} R(\widehat{M}, M)$ is relatively easier to evaluate. Its derivation details are deferred to the Appendix. In the remainder of this section we shall focus on the computation of the first gradient term, which involves partial derivatives of $\Theta' = (U, \widetilde{U}, V, \Sigma)$ with respect to malicious users $\widetilde{M}$.

We begin with the KKT condition at the optimal solution Θ' of Eq. (9.5). Unlike the alternating minimization formulation, the nuclear norm function $\|\cdot\|_*$ is not everywhere differentiable. As a result, the KKT condition relates the *subdifferential* of the nuclear norm function $\partial\|\cdot\|_*$ as

$$\mathscr{R}_{\Omega,\tilde{\Omega}}\left([M; \widetilde{M}] - [X; \widetilde{X}]\right) \in \lambda \partial \|[X; \widetilde{X}]\|_*. \tag{8.14}$$

Here $[X; \widetilde{X}]$ is the concatenated $(m+m') \times n$ matrix of X and $\widetilde{X}$. The subdifferential of the nuclear norm function $\partial\|\cdot\|_*$ is also known [178]:

$$\partial \|X\|_* = \left\{ UV^\top + W : U^\top W = WV = 0, \|W\|_2 \leq 1 \right\},$$

where $X = U\Sigma V^\top$ is the singular value decomposition of X. Suppose $\{u_i\}, \{\tilde{u}_i\}$ and $\{v_j\}$ are rows of $U, \widetilde{U}, V$ and $W = \{w_{ij}\}$. We can then re-formulate the KKT condition Eq. (9.14) as follows:

$$\begin{aligned} \forall (i,j) \in \Omega, \quad M_{ij} &= u_i^\top (\Sigma + \lambda I_\rho) v_j + \lambda w_{ij}; \\ \forall (i,j) \in \widetilde{\Omega}, \quad \widetilde{M}_{ij} &= \tilde{u}_i^\top (\Sigma + \lambda I_\rho) v_j + \lambda \tilde{w}_{ij}. \end{aligned}$$

We are now ready to derive $\nabla_{\widetilde{M}} \Theta = \nabla_{\widetilde{M}}(u, \tilde{u}, v, \sigma)$.

Evaluation of $\nabla_{\widetilde{M}} u_i$ Because u_i does not depend on $\widetilde{M}$, we have $\nabla_{\widetilde{M}} u_i = \mathbf{0}$.

Evaluation of $\nabla_{\widetilde{M}} \tilde{u}_i$ Let $\widetilde{\Omega}_i$ be all (i,j) pairs such that $(i,j) \in \widetilde{\Omega}$. Suppose we are computing the gradient of $\tilde{u}_i$ with respect to $\widetilde{M}_{i\ell}$, where ℓ can be either in or not in $\widetilde{\Omega}_i$. Define $\widetilde{\Omega}_i^\ell = \widetilde{\Omega}_i \cup \{\ell\}$ be the *extended* set of observations and denote $r = |\widetilde{\Omega}_i^\ell|$ as the size of the extended observation set. Define $\widetilde{M}_i = (\widetilde{M}_{ij})_{j \in \widetilde{\Omega}_i^\ell} \in \mathbb{R}^r$, $\tilde{w}_i = (\tilde{w}_{ij})_{j \in \widetilde{\Omega}_i^\ell} \in \mathbb{R}^r$ and $V_i^\ell = (v_j)_{j \in \widetilde{\Omega}_i^\ell} \in \mathbb{R}^{\rho \times r}$. By KKT condition,

$$\left[\left(\Sigma + \lambda I_\rho \right) V_i^\ell \right]^\top \tilde{u}_i = \widetilde{M}_i - \lambda \tilde{w}_i. \tag{8.15}$$

The above linear system can be either over-determined or under-determined, depending on the relationship between ρ and r. When the system is under-determined (e.g., $r < \rho$), the solution to Eq. (9.15) is not unique and could be instable if the matrix $A_i = \left[\left(\Sigma + \lambda I_\rho \right) V_i^\ell \right]^\top$ is ill-conditioned. On the other hand, when the system is over-determined (e.g., $r > \rho$) an exact solution $\tilde{u}_i$ may not exist. To force unique solutions in full generality, we compute $\tilde{u}_i$ by solving the following Ridge-regularized system:

$$\min_{\tilde{u}_i} \| \widetilde{M}_i - \lambda \tilde{w}_i - A_i \tilde{u}_i \|_2^2 + 2\tau \| \tilde{u}_i \|_2^2,$$

where $\tau > 0$ is a smoothing parameter. Subsequently,

$$\begin{aligned} \tilde{u}_i &\approx (A_i^\top A_i + \tau I_\rho)^{-1} A_i^\top (\widetilde{M}_i - \lambda \tilde{w}_i); \\ \frac{\partial \tilde{u}_i}{\partial \widetilde{M}_{i\ell}} &\approx (A_i^\top A_i + \tau I_\rho)^{-1} (\Sigma + \lambda I_\rho) v_\ell. \end{aligned}$$

Evaluation of $\nabla_{\widetilde{M}} v_j$ This part is similar to the gradient of $\tilde{u}_i$. Suppose we are computing $\partial v_j / \partial \widetilde{M}_{\ell j}$. Define $\bar{\Omega}_j^\ell = \Omega'_j \cup \widetilde{\Omega}'_j \cup \{\ell\}$ to be the extended set of all i such that $(i,j) \in \Omega \cup \widetilde{\Omega}$. Let $r = |\bar{\Omega}_j^\ell|$ be the size of the extended set. We then have

$$\left[(\bar{U}_i^\ell)^\top (\Sigma + \lambda I_\rho) \right] v_j = \widetilde{M}'_j - \lambda \tilde{w}'_j,$$

Algorithm 15 Optimizing $\widetilde{\boldsymbol{M}}$ via SGLD

Input: Original partially observed $m \times n$ data matrix $\boldsymbol{M}$, algorithm regularization parameter λ, attack budget parameters α, B and Λ, attacker's utility function R, step size $\{s_t\}_{t=1}^{\infty}$, tuning parameter β, number of SGLD iterations T.
Prior setup: compute $\xi_j = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{M}_{ij}$ and $\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{M}_{ij} - \xi_j)^2$ for every $j \in [n]$.
Initialization: sample $\widetilde{\boldsymbol{M}}_{ij}^{(0)} \sim \mathscr{N}(\xi_j, \sigma_j^2)$ for $i \in [m']$ and $j \in [n]$.
for $t = 0$ to T **do**
    Compute the optimal solution $\boldsymbol{\Theta}_\lambda(\widetilde{\boldsymbol{M}}^{(t)}; \boldsymbol{M})$.
    Compute gradient $\nabla_{\widetilde{\boldsymbol{M}}} R(\widehat{\boldsymbol{M}}, \boldsymbol{M})$ using Eq. (9.10).
    Update $\widetilde{\boldsymbol{M}}^{(t+1)}$ according to Eq. (9.18).
end for
Projection: find $\widetilde{\boldsymbol{M}}^* \in \arg\min_{\widetilde{\boldsymbol{M}} \in \mathbb{M}} \|\widetilde{\boldsymbol{M}} - \widetilde{\boldsymbol{M}}^{(t)}\|_F^2$. Details in the main text.
Output: $m' \times n$ malicious matrix $\widetilde{\boldsymbol{M}}^*$.

where $\bar{\boldsymbol{U}}_i^\ell$ is a $\rho \times r$ matrix consisting of all $\boldsymbol{u}_i$ or $\tilde{\boldsymbol{u}}_i$ for $i \in \bar{\Omega}_j^\ell$ as its columns. On the right-hand side, we have $\widetilde{\boldsymbol{M}}_j' = (\widetilde{\boldsymbol{M}}_{ij})_{i \in \bar{\Omega}_j^\ell}$ and $\tilde{\boldsymbol{w}}_j' = (w_{ij})_{i \in \bar{\Omega}_j^\ell}$. Let $\boldsymbol{B}_j = (\bar{\boldsymbol{U}}_i^\ell)^\top (\boldsymbol{\Sigma} + \lambda \boldsymbol{I}_\rho) \in \mathbb{R}^{r \times \rho}$ and $\tau > 0$ be a smoothing parameter. We then have

$$\frac{\partial \boldsymbol{v}_j}{\partial \widetilde{\boldsymbol{M}}_{\ell j}} \approx (\boldsymbol{B}_j^\top \boldsymbol{B}_j + \tau \boldsymbol{I}_\rho)^{-1} (\boldsymbol{\Sigma} + \lambda \boldsymbol{I}_\rho) \tilde{\boldsymbol{u}}_\ell.$$

Evaluation of $\nabla_{\widetilde{\boldsymbol{M}}} \sigma_k$ By KKT condition we have

$$\widetilde{\boldsymbol{M}}_{ij} = \tilde{\boldsymbol{u}}_{ik} \boldsymbol{v}_{jk} \cdot \sigma_k + c,$$

where c is a constant that does not depend on σ_k. Subsequently, we get

$$\frac{\partial \sigma_k}{\partial \widetilde{\boldsymbol{M}}_{ij}} = \frac{1}{\tilde{\boldsymbol{u}}_{ik} \boldsymbol{v}_{jk}}.$$

8.5.3 Mimicing Normal User Behaviors

In previous sections we initialize the malicious matrix $\widetilde{\boldsymbol{M}}$ by letting each malicious user label B items uniformly at random. This helps diversify the ratings and rated items of each malicious user to avoid being trivially detected by a simple defender. Nevertheless, using

such attack strategies the malicious users could still be relatively easily identified, because normal users generally do not rate items uniformly at random. For example, there are certain movies that are significantly more popular than the other movies or vice versa. As a result, malicious users that pick rated movies uniformly at random can be easily identified by running a t-test against a known database consisting of only normal users, as we shown in the experimental section.

To alleviate the above-mentioned issues, in this section we propose an alternative approach to compute data poisoning attacks such that the resulting malicious users $\widetilde{\boldsymbol{M}}$ mimics normal users $\boldsymbol{M}$ to avoid potential detection from the defender side, while still achieving reasonably large utility $R(\widehat{\boldsymbol{M}}, \boldsymbol{M})$ for the attacker. We use a Bayesian formulation to take both data poisoning and detection avoidance objectives into consideration. The prior distribution $p_0(\widetilde{\boldsymbol{M}})$ captures normal users' behaviors and is defined as a multivariate normal distribution

$$p_0(\widetilde{\boldsymbol{M}}) = \prod_{i=1}^{m'} \prod_{j=1}^{n} \mathcal{N}(\widetilde{\boldsymbol{M}}_{ij}; \xi_j, \sigma_j^2),$$

where ξ_j and σ_j^2 are mean and variance parameters for the rating of the jth item provided by normal users. In practice both parameters can be estimated using normal user matrix $\boldsymbol{M}$ as $\xi_j = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{M}_{ij}$ and $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{M}_{ij} - \xi_j)^2$. On the other hand, the likelihood $p(\boldsymbol{M}|\widetilde{\boldsymbol{M}})$ is defined as

$$p(\boldsymbol{M}|\widetilde{\boldsymbol{M}}) = \frac{1}{Z} \exp\left(\beta \cdot R(\widehat{\boldsymbol{M}}, \boldsymbol{M})\right), \tag{8.16}$$

where $R(\widehat{\boldsymbol{M}}, \boldsymbol{M}) = R(\widehat{\boldsymbol{M}}(\boldsymbol{\Theta}_\lambda(\widetilde{\boldsymbol{M}}; \boldsymbol{M})), \boldsymbol{M})$ is one of the attcker utility functions defined in Sec. 3.3, Z is a normalization constant and $\beta > 0$ is a tuning parameter that trades off attack performance and detection avoidance. A small β value shifts the posterior of $\widetilde{\boldsymbol{M}}$ toward its prior, which makes the resulting attack strategy less effective but harder to detect and vice versa.

Given both prior and likelihood functions, an effective detection-avoiding attack strat-

egy $\widetilde{M}$ can be obtained by sampling from its posterior distribution:

$$p(\widetilde{M}|M) = p_0(\widetilde{M})p(M|\widetilde{M})/p(M)$$
$$\propto \exp\left(-\sum_{i=1}^{m'}\sum_{j=1}^{n}\frac{(\widetilde{M}_{ij}-\xi_j)^2}{2\sigma_j^2}+\beta R(\widehat{M},M)\right). \quad (8.17)$$

Posterior sampling of Eq. (9.17) is clearly intractable, due to the implicit and complicated dependency of the estimated matrix $\widehat{M}$ on the malicious data $\widetilde{M}$, that is, $\widehat{M} = \widehat{M}(\Theta_\lambda(\widetilde{M};M)))$. To circumvent this problem, we apply *Stochastic Gradient Langevin Dynamics (SGLD, [186])* to approximately sample $\widetilde{M}$ from its posterior distribution in Eq. (9.17). More specfically, the SGLD algorithm iteratively computes a sequence of posterior samples $\{\widetilde{M}^{(t)}\}_{t\geq 0}$ and at iteration t the new sample $\widetilde{M}^{(t+1)}$ is computed as

$$\widetilde{M}^{(t+1)} = \widetilde{M}^{(t)} + \frac{s_t}{2}\left(\nabla_{\widetilde{M}}\log p(\widetilde{M}|M)\right)+\varepsilon_t, \quad (8.18)$$

where $\{s_t\}_{t\geq 0}$ are step sizes and $\varepsilon_t \sim \mathcal{N}(\mathbf{0}, s_t\boldsymbol{I})$ are independent Gaussian noises injected at each SGLD iteration. The gradient $\nabla_{\widetilde{M}}\log p(\widetilde{M}|M)$ can be computed as

$$\nabla_{\widetilde{M}}\log p(\widetilde{M}|M) = -(\widetilde{M}-\Xi)\Sigma^{-1}+\beta\nabla_{\widetilde{M}}R(\widehat{M},M),$$

where $\Sigma = \boldsymbol{diag}(\sigma_1^2,\cdots,\sigma_n^2)$ and Ξ is an $m'\times n$ matrix with $\Xi_{ij}=\xi_j$ for $i\in[m']$ and $j\in[n]$. The other gradient $\nabla_{\widetilde{M}}R(\widehat{M},M)$ can be computed using the same procedure listed in previous sections 9.5.1 and 9.5.2. Finally, the sampled malicious matrix $\widetilde{M}^{(t)}$ is projected back onto the feasible set $\mathbb{M}$ by selecting B items per user with the largest absolute rating and truncating ratings to the level of $\{\pm\Lambda\}$. A high-level description of the proposed method is given in Algorithm 15.

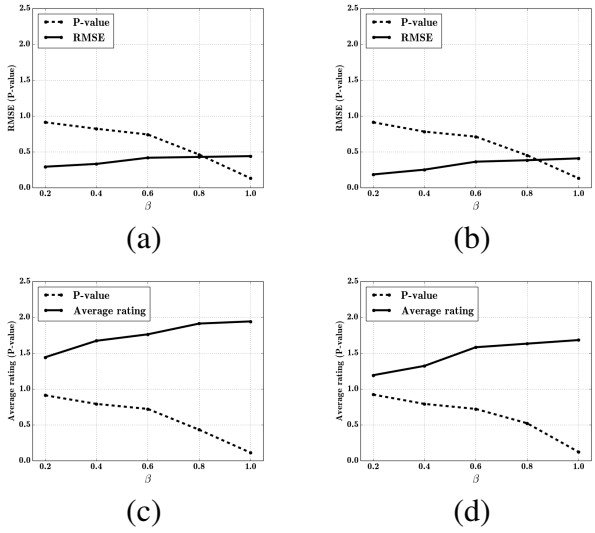


Figure 8.2: P values and RMSE/Averge ratings for alternating minimization with different β values; (a) $\mu_1 = 1, \mu_2 = 0$, (b) $\mu_1 = 1, \mu_2 = -1$, (c) $\mu_1 = 0, \mu_2 = 1$, (d) $\mu_1 = -1, \mu_2 = 1$.

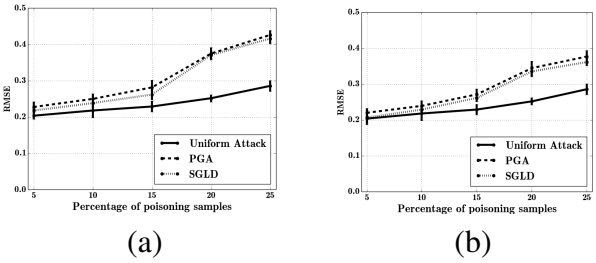
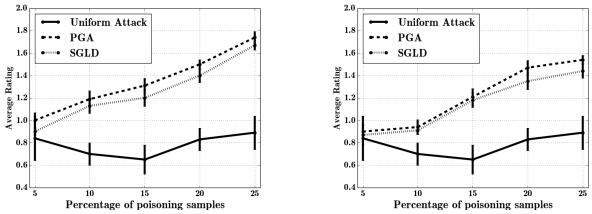


Figure 8.3: RMSE for alternating minimization with different percentage of malicious profiles; (a) $\mu_1 = 1, \mu_2 = 0$, (b) $\mu_1 = 1, \mu_2 = -1$.

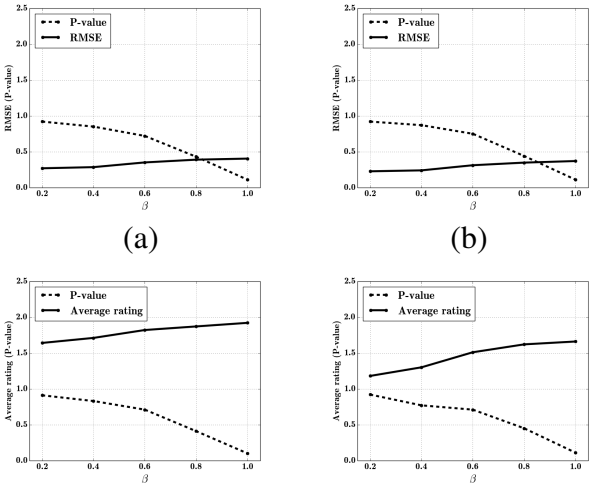
8.6 Results

To evaluate the effectiveness of our proposed poisoning attack strategy, we use the publicly available MovieLens dataset which contains 20 millions ratings and 465,000 tag applications applied to 27,000 movies by 138,000 users [196]. We test data poisoning attacks on both alternating minimization based and nuclear norm minimization based collaborative filtering systems. All ratings are integer valued between one (most disliked) and five (most liked). We shift the rating range to $[-2, 2]$ for computation convenience. To avoid the "cold-start" problem, we consider users who have rated at least 20 movies. Two met-



(a) (b)

Figure 8.4: Average ratings of certain items using alternating minimization; (a) $\mu_1 = 0, \mu_2 = 1$, (b) $\mu_1 = -1, \mu_2 = 1$.



(a) (b)

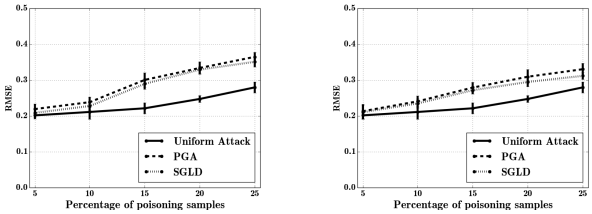
(c) (d)

Figure 8.5: P values and RMSE/Averge ratings for nuclear norm minimization with different β values; (a) $\mu_1 = 1, \mu_2 = 0$, (b) $\mu_1 = 1, \mu_2 = -1$, (c) $\mu_1 = 0, \mu_2 = 1$, (d) $\mu_1 = -1, \mu_2 = 1$.

rics: root mean square error (RMSE) for the predicted unseen entries [2] and average rating for specific items, are employed to measure the relative performance of the systems before and after data poisoning attacks. Values of these metrics are plotted against percentage of malicious profiles of the total number of normal profiles in the system.

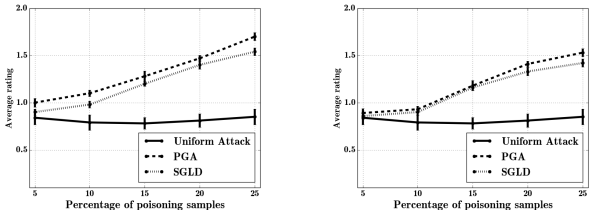
We first analyze the tradeoff between attack performance and detection avoidance, which is controled by the β parameter in Eq. (9.16). This serves as guidence of how β should be set in later experiments. We use paired t-test to compare the distributions of rated items between normal and malicious users. Figure 9.2 plots P-values and RMSE/Av-

[2] defined as RMSE $= \sqrt{\sum_{(i,j)\in\Omega^C} (\overline{\boldsymbol{M}}_{ij} - \widehat{\boldsymbol{M}}_{ij})^2 / |\Omega^C|}$, where $\overline{\boldsymbol{M}}$ is the prediction of model trained on clean data $\mathscr{R}_\Omega(\boldsymbol{M})$ only (i.e., without data poisoning attacks).



(a) (b)

Figure 8.6: RMSE for nuclear norm minimization with different percentage of malicious profiles; (a) $\mu_1 = 1, \mu_2 = 0$, (b) $\mu_1 = 1, \mu_2 = -1$.



(a) (b)

Figure 8.7: Average ratings of certain items using nuclear norm miminization; (a) $\mu_1 = 0, \mu_2 = 1$, (b) $\mu_1 = -1, \mu_2 = 1$.

erage ratings against different values of β. When $B = 25$ (recall that B is the maximum number of items a malicious user is allowed to rate), with the increase of β, the P-value decreases while both RMSE and average per-item ratings increase. To strive for a good tradeoff, we set $\beta = 0.6$ at which the P-value stablizes around 0.7 and the poisoning attack performance is not much sacrificed.

We employ attack models specified in Eq. (9.9), where the utility parameter μ_1 and μ_2 balance two different malicious goals (availability and integrity) an attacker wishes to achieve. For the integrity utility $R^{\text{in}}_{J_0,w}$, the J_0 set contains only one item j_0 selected randomly from all items whose average predicted ratings are around 0.8. The weight w_{j_0} is set as $w_{j_0} = 2$. Figure 9.3 plots the RMSE after data poisoning attacks. When $\mu_1 = 1$, $\mu_2 = 0$, the attacker is interested in increasing the RMSE of the collaborative filtering system and hence reducing the system's availability. On the other hand, when $\mu_1 = 1$, $\mu_2 = -1$ the attacker wishes to increase RMSE while at the same time keeping the rating

of specific items (j_0) as low as possible for certain malicious purposes. [3] Figure 9.3 shows that when the attackers consider to both objectives ($\mu_1 = 1, \mu_2 = -1$), the RMSE after poisoning is slightly lower than that if only availability is targeted ($\mu_1 = 1, \mu_2 = 0$). In addition, the *projective gradient ascent* (PGA) strategy generates the largest RMSE score compared with the other methods. However, PGA requires malicious users to rate each item uniformly at random, which might expose the malicious profiles to an informed defender. More specifically, the paired t-test on those malicious profiles produced by PGA rejects the null hypothesis that the items rated by the attacker strategies are the same as those obtained from normal users ($p < 0.05$). In contrast, the *stochastic gradient langevin dynamics (SGLD)* method leads to slightly worse attacker utility but generates malicious users that are hard to distinguish from the normal users (for example, the paired t-test leads to inconclusive P values (larger than 0.7) with $\beta = 0.6$. Finally, both PGA and SGLD result in higher attacker utility compared to *uniform attacks*, where both ratings and rated items are sampled uniformly at random for malicious profiles.

Apart from the RMSE scores, we also plot ratings of specific items against percentage of malicious profiles in Figure 9.4. We consider two additional attack utility settings: $\mu_1 = 0, \mu_2 = 1$, in which the attacker wishes to push the ratings of some particular items (specified in w and J_0 of R^{in}) as high as possible; and $\mu_1 = -1, \mu_2 = 1$, where the attacker also wants to leave a "light trace" by reducing the impact on the entire system resulted from malicious activities. Figure 9.4 shows that targeted attackes (both PGA and SGLD) are indeed more effective in terms of manipulating ratings of specific items for integrity attacks.

We also plot RMSE/Average ratings against malicious user percentage in Figure 9.5, 9.6 and 9.7 for the nuclear norm minimization formulation under similar settings. Because nuclear norm mininmization is more computationally expensive than alternating minimization, we uniformly select a random subset of Movielens that consists of 1000 users and

[3] Due to space limits, average ratings for the setting $\mu_1 = 1, \mu_2 = -1$ are plotted in Appendix.

1700 movies (items) in our experiments. To set penalty and smoothness parameters λ and τ, we sub-sample 100 users and 150 movies uniformly at random and tune these parameters so that the RMSE is minimized on the small subsampled data set. The parameters are set as $\lambda = 0.1$ and $\tau = 0.01$ through this procedure. The "detection-avoidance" parameter β is again set to 0.6 according to the plots of P values and RMSE depicted in Figure 9.5. In general, we observe similar behavior of both RMSE/Average ratings under different attakcing models μ_1, μ_2 as in previous figures (9.2, 9.3 and 9.4) for alternating minimization.

8.7 Summary of Contributions

The poisoning attack presented in this work is the first step toward the security analysis of collaborative filters against poisoning attacks. Our ultimate goal is to come up with possible defensive strategies based on the careful analysis of adversarial behaviors. Since the poisoning data is optimized based on the attacker's malicious objectives, the correlations among features within a feature vector may change to appear different with the normal instances. Therefore, tracking and detecting deviations in the feature correlations and other different accuracy metrics can be one potential defense. For example, the defender can periodically construct a model using the training dataset, evaluate its accuracy on the manually selected validation dataset, and raise an alarm in case of any suspicious change in the accuracy metrics. These metrics can be the number of correctly classified instances, or the Kappa statistic, which measures relative improvement over random predictors. Additionally, defender can also apply the combinational models or sampling strategies, such as bagging, to reduce the influence of the poisoning data during training if the poisoning data can be viewed as a particular category of outliers to counter the poisoning attacks.

Chapter 9

HIDING SENSITIVE DATA IN PLAIN SIGHT: ITERTIVE CLASSIFICATION FOR SANITIZING LARGE-SCALE DATASETS

Another anchor of *secure learning* is to protect data privacy, where adversaries can undermine the integrity of the learner and therefore compromise the privacy preserving systems. In this chapter we discuss the potential attacks targeting on data privacy and provide a flexible solution to balance the data privacy and utility for data publishing. The data deluge enabled by the widespread use of information technology, and the sensitive information such data often contains, present a dilemma: sharing the data can lead to significant new discoveries, but may also leak sensitive information. Such concerns are salient in the context of electronic medical records, clinical trial data, classified data, social media data, and many other domains. When dealing with unstructured data, such as text, sanitizing it to remove sensitive information is challenging at scale, since sensitive content, such as patient names, is not usually labeled as such. A natural approach is to add sensitivity labels to a small subset of entities (e.g., words), use classification learning to predict labels on the residual data, remove predicted sensitive information, and release the remainder. However, using an imperfect classifier will inevitably lead one to leak sensitive information. We model this problem as a game between a publisher who chooses a set of classifiers to apply to data, releasing only predicted negatives, and an attacker who chooses a classifier to prioritize a list of entities to inspect, subject to an inspection budget constraint. We show that if the loss from attack success is high, all locally optimal publishing policies ensure that the attacker's classifier uncovers few true positives, and a high-budget attacker can do little better than choosing entities uniformly at random. In addition, we exhibit a greedy algorithm which will converge to such a local optimum within a linear number of iterations, and analyze sample complexity of this algorithm. We also demonstrate the effectiveness at

sanitizing data of our approach by extensive experimental results.

9.1 Overview

Vast quantities of personal health data are now collected in a wide variety of settings. It is anticipated that such data can enable significant improvements in the quality of health services provided to individuals and facilitate new discoveries for society. At the same time, the data collected is often sensitive, and regulations, such as the Privacy Rule of the Health Insurance Portability and Accountability Act of 1996 (when disclosing medical records) [197] and the European Data Protection Directive [198] often recommend the removal of identifying information. To accomplish such goals, the past several decades have brought forth the development of numerous data protection models [199]. These models invoke various principles, such as hiding individuals in a crowd (e.g., k-anonymity [31]) or perturbing values to ensure that little can be inferred about an individual even with arbitrary side information (e.g., ε-differential privacy [200]). All of these approaches are predicated on the assumption that the publisher of the data knows where the identifiers are from the outset. More specifically, they assume the data has an explicit representation, such as a relational form [201], where the data has at most a small set of values per feature [202, 203, 204, 205].

However, it is increasingly the case that the data we generate lacks a formal relational (or explicitly structured) representation. A clear example of this phenomenon is the substantial quantity of natural language text which is created in the clinical notes in medical records [206]. To protect such data, there has been a significant amount of research into natural language processing (NLP) techniques to detect (and subsequently redact or substitute) identifiers [207, 208, 209, 210]. As demonstrated through systematic reviews [62] and various competitions [211, 212], the most scalable versions of such techniques are rooted in, or rely heavily upon, machine learning methods, in which the publisher of the data annotates instances of personal identifiers in the text, such as patient and doctor name, social security

number, and a date of birth, and the machine attempts to learn a classifier (e.g., a grammar) to predict where such identifiers reside in a much larger corpus. Unfortunately, no learned classifier is perfect, and some sensitive information will invariably leak through to the data recipient. This is clearly a problem if, for instance, the information leaked corresponds to direct identifiers (e.g., personal name) or quasi-identifiers (e.g., ZIP codes or dates of birth) which may be exploited in re-identification attacks, such as the re-identification of Thelma Arnold in the search logs disclosed by AOL [213] or the Social Security Numbers in Jeb Bush's emails [214].

Rather than attempt to detect and redact every sensitive piece of information, our goal is to guarantee that even if identifiers remain in the published data, the adversary cannot easily find them. Fundamental to our approach is the acceptance of non-zero privacy risk, which we view as unavoidable. This is consistent with most privacy regulation, such as HIPAA, which allows expert determination that privacy "risk is very small" [197], and the EU Data Protection Directive, which "does not require anonymisation to be completely risk-free" [215]. Our starting point is a threat model within which an attacker uses published data to first train a classifier to predict sensitive entities (based on a labeled subset of the data), prioritizes inspection based on the predicted positives, and inspects (and verifies the true sensitivity status of) B of these in a prioritized order. Here, B is the budget available to inspect (or read) instances and *true sentivie* entities are those which have been correctly labeled as sensitive (for example, *true sensitive* entities could include identifiers such as a name, social security number, and address). An illustration of such a setting is depicted in Figure 10.1. In this threat model, we consider an idealized adversary with several elements of omniscience. First, we assume that the adversary can always correctly assess the true sensitivity for any manually inspected instance. Second, we assume that the adversary computes an optimal classifier (that is, a classifier with maximum accuracy within a given hypothesis class) with respect to published data.

We use this threat model to construct a game between a *publisher*, who 1) applies a

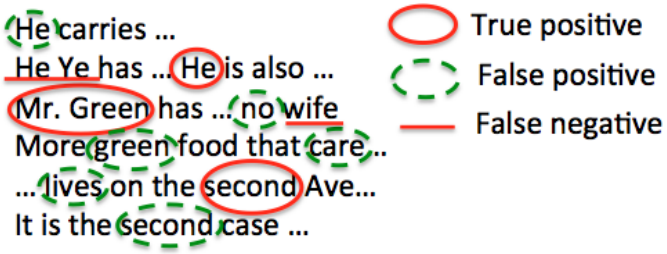


Figure 9.1: An example of sensitive and non-sensitive instances that need to be distinguished via manual inspection.

collection of classifiers to an original data set, 2) prunes all the positives predicted by any classifier, and 3) publishes the remainder, and an *adversary* acting according to our threat model. We show that any locally optimal publishing strategy exhibits the following two properties when the risk associated with exploited personal identifiers is high: *a*) an adversary cannot learn a classifier with a high true positive count, and *b*) an adversary with a large inspection budget cannot do much better than manually inspecting and confirming instances chosen uniformly at random (i.e., the classifier adds little value). When these conditions hold, we say that sensitive data is *hiding in plain sight*—even though it may be leaked, it is difficult for a motivated adversary to discover. Moreover, we exhibit a greedy publishing strategy which is guaranteed to converge to a local optimum (and consequently guarantees the above two properties) in a linear (in the size of the data) number of iterations. Our experiments on two distinct electronic health records data sets demonstrate the power of our approach, showing that 1) the number of residual true positives is always quite small, 2) confirming that the attacker with a large budget cannot do much better than uniformly randomly choosing entities to manually inspect, 3) demonstrating that most ($> 93\%$) of the original data is nevertheless published, and 4) showing that in practice the number of required algorithm iterations (< 5) is a small fraction of the size of the data. Additional experiments, involving two non-health-related datasets corroborate these findings, demonstrating generalizability of this approach. Figure 10.2 illustrates the general attack strategy on privacy from the perspective of secure learning.

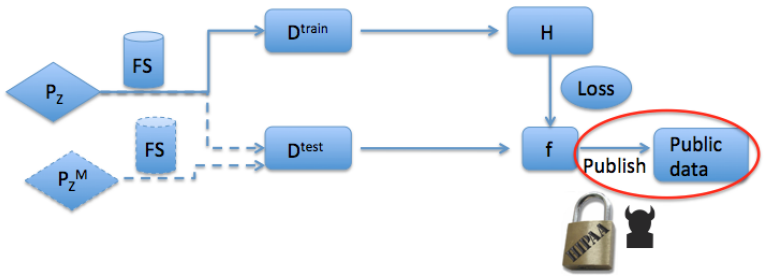


Figure 9.2: General idea of the exploratory privacy preserving attacks

9.2 Related Work

9.2.1 Approaches for Anonymizing Structured Data

There has been a substantial amount of research conducted in the field of privacy-preserving data publishing (PPDP) over the past several decades [216, 199]. Much of this work is dedicated to methods that transform well-structured (e.g., relational) data to adhere to a certain criterion (or set of criterion), such as k-anonymization [31], l-diversity [56], m-invariance [60], and ε-differential privacy [200], among a multitude of others. These criteria attempt to offer guarantees about the ability of an attacker to either distinguish between different records in the data or make inferences tied to a specific individual. There is now an extensive literature aiming to operationalize such PPDP criteria in practice through the application of techniques such as generalization, suppression (or removal), and randomization (e.g., [66, 217, 218, 70, 71, 74]). All of these techniques, however, rely on *a priori* knowledge of which features of the data are either themselves sensitive or can be linked do sensitive attributes. This is a key distinction from our work: we aim to *automatically discover* which entities in unstructured data are sensitive, as well as ensure (in a formal sense) that whatever sensitive data remains cannot be easily unearthed by an adversary.

9.2.2 Sanitizing Unstructure Data

In the context of privacy preservation for unstructured data, such as text, various approaches have been proposed for the automatic discovery of sensitive entities, such as identifiers. The simplest of these rely on a large collection of rules, dictionaries, and regular expressions (e.g., [219, 220]). [221] proposed an automated data sanitization algorithm aimed at removing sensitive identifiers while inducing the least distortion to the contents of documents. However, this algorithm assumes that sensitive entities, as well as any possible related entities, have already been labeled. Similarly, [222] have developed the t-plausibility algorithm to replace the known (labeled) sensitive identifiers within the documents and guarantee that the sanitized document is associated with least t documents.

9.2.3 Machine Learning Methods for Sanitizing Unstructured Data

A key challenge in unstructured data that makes it qualitatively distinct from structured is that even identifying (labeling) which entities are sensitive is non-trivial. For example, while a structured portion of electronic medical records would generally have known sensitive categories, such as a patient's name, physician's notes do not have such labels, even though they may well refer to a patient's name, date of birth, and other potentially identifying information. While rule-based approaches, such as regular expressions, can automatically identify some of the sensitive entities, they have to be manually tuned to specific classes of data, and don't generalize well. A natural idea, which has received considerable traction in prior literature, is to use machine learning algorithms, trained on a small portion of labeled data, to automatically identify sensitive entities. Numerous classification algorithms have been proposed for this purpose, including decision stumps [223], support vector machines (SVM) [224], conditional random fields (CRFs) [207, 210, 225], hybrid strategies that rely on rules and statistical learning models [226, 227] ensemble methods [62]. Unfortunately, all of such PPDP algorithms fail to formally consider the

adversarial model, which is crucial for the decision making of the data publisher.

Therefore, our approach builds on this literature, but is quite distinct from it in several ways. First, we propose a novel explicit threat model for this problem, allowing us to make formal guarantees about the vulnerability of the published data to adversarial re-identification attempts. Second, we introduce a natural approach for sanitizing data that uses machine learning in an iterative framework. Notably, this approach performs significantly better than a standard application of CRFs, which is the leading approach for text sanitization to date [228], but can actually make use of arbitrary machine learning algorithms. Our work can be seen within the broader context of game theoretic modeling of security and privacy [169, 229, 140, 230, 231], including a number of efforts that use game theory to make machine learning algorithms robust in adversarial environments [232, 174, 233, 115, 113, 114]. In both of these genres of work, a central element is an explicit formal threat (i.e., attacker) model, with the game theoretic analysis generally focused on computing defensive privacy-preserving strategies. None of this work to date, however, addresses the problem of PPDP of unstructured data with sensitive entities not known *a priori*.

9.3 Model

Before delving into the technical details, we offer a brief high-level intuition behind the main idea in this study. Suppose that a publisher uses a machine learning algorithm to identify sensitive instances in a corpus, these instances are then redacted, and the residual data is shared with an attacker. The latter, aspiring to uncover residual sensitive instances (e.g., identifiers) can, similarly, train a learning algorithm to do so (using, for example, a subset of published data that is manually labeled). Now, to be a bit crude, consider two possibilities: first, the learning algorithm enables the attacker to uncover a non-trivial amount of sensitive information, and second, the learning algorithm is relatively unhelpful in doing so. In the latter case, the publisher can perhaps breath freely: few sensitive entities

can be identified by this attacker, and the risk of published data is low. The former case is, of course, the problem. However, notice that, in principle, the publisher can *try out* this attack in advance of publishing the data, to see whether it can in fact succeed in this fashion. Moreover, if the attacker is projected to be sufficiently successful, the publisher has a great deal to gain by *redacting the sensitive entities an attacker would have found*. Of course, there is no need to stop at this point: the publisher can keep simulating attacks on the published data, and redacting data labeled as sensitive, until these simulations suggest that the risk is sufficiently low. This, indeed, is the main idea. However, many details are clearly missing: for example, what does an attacker do after training the learning algorithm, when, precisely, should the publisher stop, and what can we say about the privacy risk if data is published in this manner, under this threat model? Next, we formalize this idea, and offer precise answers to these and other relevant questions.

Table 10.1 summarizes all the notation used. Imagine that a publisher's dataset consists of a set of n entities (or words), $X = \{x_1, \ldots, x_n\}$, of which he will publish a subset $P \subseteq X$. The publisher may have an additional data set for training a classifier to predict whether an entity x is sensitive. We let α denote the fraction of the original n entities that are sensitive. A learning algorithm is designed to select a hypothesis that best supports the data. Here we consider the hypothesis to be a function f mapping from the data space $\mathscr{D}$ to the response space $\mathscr{E}$; i.e., $f : \mathscr{D} \to \mathscr{E}$. Of course there are many such hypotheses. We assume f belongs to a family of hypotheses $\mathscr{H}$. Specifically, the response space $\mathscr{E} = \{0, 1\}$ within our problem indicates whether the entity x is sensitive (S, $f(x) = 1$) or non-sensitive (N, $f(x) = 0$), and $\mathscr{H}$ represents a set of binary classifiers. A crucial assumption in our approach is that the hypothesis class $\mathscr{H}$ is known to the public, including the publisher and attackers. This is a natural assumption, considering that state-of-the-art machine learning algorithms are well known and typically have multiple high-quality open source implementations. Moreover, even as new approaches are developed for identifying sensitive entities in unstructured (e.g., text) data, these approaches can be subsequently incorporated into

Table 9.1: Table of Notations

n	$\triangleq$	number of total instances
$\mathscr{H}$	$\triangleq$	hypothesis class of the publisher
H	$\triangleq$	the subset of classifiers chosen by the publisher
S	$\triangleq$	sensitive instances
N	$\triangleq$	non-sensitive instances
$TP(h,P)$	$\triangleq$	number of *true positives* by h on P
$TN(h,P)$	$\triangleq$	number of *true negatives* by h on P
$FP(h,P)$	$\triangleq$	number of *false positives* by h on P
$FN(h,P)$	$\triangleq$	number of *false negatives* by h on P
TP_A	$\triangleq$	number of *true positives* obtained by attacker
TN_A	$\triangleq$	number of *true negatives* obtained by attacker
FP_A	$\triangleq$	number of *false positives* obtained by attacker
FN_A	$\triangleq$	number of *false negatives* obtained by attacker
TP_D	$\triangleq$	number of *true positives* obtained by defender
TN_D	$\triangleq$	number of *true negatives* obtained by defender
FP_D	$\triangleq$	number of *false positives* obtained by defender
FN_D	$\triangleq$	number of *false negatives* obtained by defender
α	$\triangleq$	percent of identifiers in data
h_A	$\triangleq$	the attacker's classifier
$T(H)$	$\triangleq$	loss function of data publisher for H

our framework. Note that our assumption of common knowledge of $\mathscr{H}$ does *not* imply that the publisher knows the actual function f used by the attacker (see threat model below); the importance of this point is highlighted when we analyze finite sample bounds in Section 10.5.

We use h to denote a classifier chosen from the hypothesis class $\mathscr{H}$. For a classifier h

and a data set Y, we introduce the following notation:

- $FP(h,Y) = |\cup_{x\in Y}\{x \in N | h(x) = 1\}|$: the number of false positive instances of h on Y,
- $TP(h,Y) = |\cup_{x\in Y}\{x \in P | h(x) = 1\}|$: the number of true positive instances of h on Y,
- $FN(h,Y) = |\cup_{x\in Y}\{x \in P | h(x) = 0\}|$: the number of false negative instances of h on Y, and
- $TN(h,Y) = |\cup_{x\in Y}\{x \in N | h(x) = 0\}|$: the number of true negative instances of h on Y.

Clearly, if $|Y| = m$, $FP(h,Y) + TP(h,Y) + FN(h,Y) + TN(h,Y) = m\ \forall h \in \mathscr{H}$. Finally, we define $FP(h,\emptyset) = FN(h,\emptyset) = TP(h,\emptyset) = TN(h,\emptyset) \equiv 0$.

9.3.1 Threat Model

Suppose that an adversary obtains the published data $P \subseteq X$. We assume that an adversary has a fixed inspection budget, B, which can be thought of as manual inspection of actual instances to verify whether or not they are sensitive (and, consequently, have value to the adversary). If a sensitive instance is found, we assume that it is exploited by an adversary, who gains L for every such instance, which is identical to the publisher's loss. Thus, when the attacker selects a set $I \subseteq P$ of instances for inspection, such that $|I| \leq B$, his utility is

$$U_A(I) = L|\{\text{sensitive instances} \in I\}| = L\sum_{x\in I} S(x), \tag{9.1}$$

where $S(x) = 1$ iff x is sensitive. A central aspect of the threat model is the specific way that the attacker chooses the set I of instances to inspect. A simple baseline is to choose I uniformly at random from P. We use U_A to denote the utility that the attacker obtains when using this simple baseline. Presumably, however, the attacker can do better by using a more

sophisticated strategy. In particular, we suppose that a *sophisticated* attacker proceeds as follows:

1. Choose a classifier

$$h_A(P) \in \arg\min_{h \in \mathscr{H}} \frac{FP(h,P) + FN(h,P)}{|P|}. \tag{9.2}$$

   In other words, the attacker chooses an optimal classifier from $\mathscr{H}$ in terms of accuracy. From the publisher's perspective, this is a very pessimistic limit of an attacker who uses a subset of P for training a standard classification algorithm, such as an SVM.

2. Prioritize instances in P by ranking all $x \in P$ with $h^*(x) = 1$ first, followed by those with $h^*(x) = 0$. Within each class, the order is arbitrary.

3. Choose I in this ranked order until it contains B instances. In other words, first the attacker will choose the predicted positives, followed by predicted negatives (if there is any budget remaining).

We simply refer to h_A where P is clear from context. We let U_A^* denote the attacker's utility when using this more sophisticated learning-based strategy. A technical caveat is that depending on the quality of the classifier, U_A^* is not necessarily higher than U_A; below, we provide a sufficient condition for $U_A^* \geq U_A$.

As an illustration, let us return to Figure 10.1 which presents an example of the behavior of an attacker given a published dataset containing sensitive and non-sensitive instances. Assume the circled words are classified as positives by h_A. Therefore, the attacker would inspect these words and their surrounding context first. However, in this setting, some of the words inspected are not sensitive instances (i.e., false positives; shown in dashed ovals). For example, the first dashed "He" is a pronoun, while the solid circled "He" is actually the name of a person. Therefore, if the attacker has sufficient budget to inspect all of the

circled instances, he would gain 3 units of utility (i.e., true positives, shown in solid ovals), and waste 3 units of budget (again, in dashed ovals).

9.3.2 Data Publisher Model

To develop some intuition for our publisher model, let us first consider the typical approach for sanitizing data (we assume for now that the defender is able to learn an optimal classifier; we relax this assumption below):

1. Learn a classifier
$$\bar{h} \in \arg\min_{h \in \mathscr{H}} \frac{FP(h,X) + FN(h,X)}{|X|}. \tag{9.3}$$
Let $X_1 = \{x \in X | \bar{h}(x) = 1\}$ (i.e., X_1 is the set of predicted positives).

2. Publish the data set $P = X \setminus X_1$.

Essentially all of the approaches in the literature assume this, or a similar, form. To apply our threat model above, we consider two possibilities: a) the attacker's classifier h_A can successfully identify residual sensitive instances, or b) the attacker's classifier cannot detect residual positives. If we are in situation (b), the publisher can view the sanitization as a success. Situation (a), on the other hand, is clearly problematic, but it also suggests a natural solution: the publisher can apply h_A to residual data, remove the sensitive instances, and only then publish the data. Indeed, this is where the symmetry between the publisher and attacker, taking advantage of the common knowledge of $\mathscr{H}$, is pivotal. Specifically, *the publisher can simulate anything that the attacker would do*.

Moreover, there is no reason to stop at this point. In fact, the publisher should continue as long as the simulated classifier that would be used by the attacker is sufficiently good. This observation also offers the key intuition for our results. Whenever the publisher chooses to stop, the attacker's ability to identify sensitive instances must inherently be relatively weak. Of course, this will depend on the relative loss to the publisher from correctly identified sensitive entities and the value of publishing data.

Using the developed intuition, we model the publisher as selecting a finite set of classifiers $H \subseteq \mathscr{H}$, where $H = \{h_1, h_2, ..., h_D\}$. Figure 10.3 shows the process of generating and publishing the data in Figure 10.1. After applying each classifier h_i, the positive instances are replaced with the fake tokens, such as "**[NAME]**" replacing an individual's name.

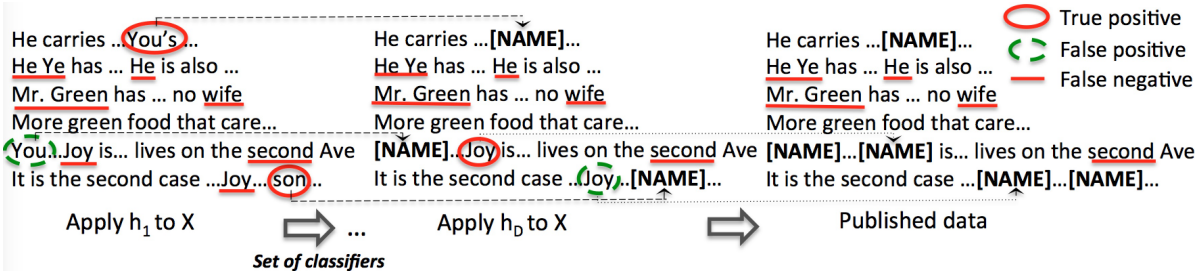


Figure 9.3: The process for applying a set of classifiers H to data X.

Let $X_1(H) = \cup_{h \in H}\{x \in X | h(x) = 1\}$, that is, the set of all positives predicted by the classifiers in H, and let $P(H) = X \setminus X_1(H)$; we use P with no argument where H is clear from context. The publisher's approach is:

1. Choose a collection of classifiers H (we address this choice below).

2. Publish the data set $P(H) = X \setminus X_1(H)$.

Let $FN(H)$ be the number of false negatives of H in X, which we define as all residual sensitive instances in P, and let $FP(H)$ be the number of false positives in X, that is, all predictive positives by any $h \in H$ which are, in fact, not sensitive. It is immediate that for any H, $FN(H) \leq \alpha n$ (the number of false negatives is at most the total number of sensitive entities in the original data) and $TN(H) \leq (1-\alpha)n$ (the number of true negatives is at most the total number of non-sensitive entities). If we allow the attacker to have an infinite budget, then every false negative will be exploited, resulting in the total loss of $L \cdot FN(H)$. In addition, each false positive costs the publisher a fixed amount C, which we can interpret as the value of publishing the data. Thus, we define the (worst-case) total loss

to the publisher from using a set of classifiers H as

$$T(H) = L \cdot FN(H) + C \cdot FP(H), \tag{9.4}$$

where $FN(H) = |\cap_{h \in H} \{x \in S | h(x) = 0\}|$ and $FP(H) = |\cup_{h \in H} \{x \in N | h(x) = 1\}|$, where S, N represent the sensitive and non-sensitive instances, respectively. $TN(H)$ and $TP(H)$ can be defined similarly.

9.3.3 Contextual Information and Inference Attacks

A significant amount of work in privacy and data sanitization deals with linkage attacks. Of particular relevance to our purpose are correlations among words in documents which enable an attacker to recover some sensitive information that has been removed [209]. Our methods can be extended directly to consider contextual information in two ways. First, we can use previous methods to discover entities in training data correlated with identifiers, and label these as identifiers as well. We can then apply our methods separately for different categories of identifiers as well as derived (correlated) words and phrases to remove both identifying information and any contextual data. Alternatively, we can first apply our methods to learn a collection of classifiers predicting identifiers in test data, and use association-based methods, such as [209], to remove additional contextual information from the test data. Henceforth, we focus on the core problem of predicting identifiers.

9.4 A Greedy Algorithm for Automated Data Sanitization

Given a formal model, we can now present our iterative algorithm for automated data sanitization, which we term *GreedySanitize*. Our algorithm (shown as Algorithm 16) is simple to implement and involves iterating over the following steps: 1) compute a classifier on training data, 2) remove all predicted positives from the training data, and 3) add this classifier to the collection. The algorithm continues until a specified stopping condition is

satisfied, at which point we publish only the predicted negatives, as above. It is important

Algorithm 16 GreedySanitize(X_t), X_t : training data.

$H \leftarrow \{\}$, $k \leftarrow 0$, $h_0 \leftarrow \emptyset$, $D_0 \leftarrow X_t$,
repeat
  $H \leftarrow H \cup h_k$
  $k = k + 1$
  $h_k \leftarrow$LearnClassifier(D_{k-1})
  $D_k \leftarrow$RemovePredictedPositives(D_{k-1}, h_k)
until $T(H \cup h_k) - T(H) \geq 0$
return H

to emphasize that *GreedySanitize* is *qualitatively different* from typical ensemble learning schemes in several ways. First, it is a crucial feature of the algorithm that a classifier is re-trained in every iteration on data which includes only predicted negatives from all prior iterations; this is entirely unlike the mechanics of any ensemble learning algorithm we are aware of.[1] Second, our algorithm removes the union of all predicted positives, whereas ensemble learning typically applies a weighted voting scheme to predict positives; our algorithm, therefore, is fundamentally more conservative when it comes to sensitive entities in the data. Third, the stopping condition is uniquely tailored to the algorithm, and is critical in enabling us to provide provable guarantees about privacy-related performance of the algorithm.

Given the iterative nature of the algorithm, it is not obvious that it is always guaranteed to terminate. The following theorem asserts that *GreedySanitize* will always terminate in a linear number of iterations.

Theorem 9.4.1. *Algorithm 16 terminates after at most $|X_t|$ iterations.*

Proof. Let $TP(D)$, $FP(D)$, $TN(D)$, and $FN(D)$ specifically refer to these quantities computed on *training data D*. Suppose that there exists an iteration i such that $TP(D_{i-1}) = 0$. It is clear that Algorithm 16 will stop after this iteration. Now, suppose instead that

[1] Typical ensemble learning algorithms will either focus on mistakes made in prior iterations (boosting is an example of this), take no note of performance by other members of the ensemble (e.g., bagging), or use a fixed set of classifiers as inputs into a meta-classifier [234].

$TP(D_{i-1}) \geq 1$ in every iteration. In this case, in at most n iterations no data will remain, and $TP(\emptyset) = 0$ by definition. Consequently, either $TP(D_{i-1}) = 0$ for $i < n$ and the algorithm will terminate, or the algorithm will stop when $i = n$. □

9.5 Analysis of *GreedySanitize*

Our theoretical analysis of the proposed *GreedySanitize* algorithm focuses on two questions: first, what kinds of privacy guarantees does this algorithm offer, and second, how to generalize the privacy guarantees to account for finite sample approximations inherent in the algorithm. To address the first question, we abstract away the details of our algorithm behind the veil of its stopping condition, which turns out to be the primary driver of our results. This also allows us to state the privacy guarantees in much more general terms.

9.5.1 Analysis of Locally Optimal Publishing Policies

In this section we analyze the adversary's ability to infer sensitive information from published data if the defender's choice of classifiers H to apply to original data satisfies the following *local optimality* condition.

Definition 9.5.1. *A set of classifiers $H \subseteq \mathscr{H}$ is a* local optimum *if $T(H \cup h_A) - T(H) \geq 0$.*

In plain terms, a subset is a local optimum if the adversary's optimal classifier h_A (that is, the attacker's best classifier choice to apply to the published data), when added to this subset, does not improve the publisher's utility. Under a minor regularity condition that $\mathscr{H}$ contains an identity (which can always be added), there is always a trivial local optimum of not releasing any data. Notice that the local optimality condition is exactly the stopping condition of *GreedySanitize*, which means that when the algorithm terminates, its output set of hypotheses H is guaranteed to be a local optimum.

We now present a lemma that enables us to characterize *all of the local optima*.

Lemma 9.5.1. *For an arbitrary set of classifiers $H \subseteq \mathscr{H}$,*

1. $FN(H) = FN(H \cup h) + TP(h, P(H))$, *and*
2. $FP(H \cup h) = FP(H) + FP(h, P(H))$.

Proof. For result 1, define the set

$$\widetilde{FN}(H) = \cap_{\tilde{h} \in H} \{x \in S | \tilde{h}(x) = 0\}. \tag{9.5}$$

Thus,

$$\widetilde{FN}(H \cup h) = \cap_{\tilde{h} \in H} \{x \in S | \tilde{h}(x) = 0\} \cap \{x \in S | h(x) = 0\}. \tag{9.6}$$

We can represent $\widetilde{FN}(H)$ as

$$\begin{aligned} \widetilde{FN}(H) =& (\widetilde{FN}(H) \cap \{x \in S | h(x) = 0\}) \\ & \cup (\widetilde{FN}(H) \cap \{x \in S | h(x) = 1\}) \\ =& \widetilde{FN}(H \cup h) \cup (\widetilde{FN}(H) \cap \{x \in S | h(x) = 1\}). \end{aligned} \tag{9.7}$$

Moreover, note that $x \in \widetilde{FN}(H)$ implies that $x \in P(H)$, so that

$$\begin{aligned} \widetilde{FN}(H) =& \widetilde{FN}(H \cup h) \cup (\widetilde{FN}(H) \\ & \cap \{x \in P(H) \cap S | h(x) = 1\}) \\ =& \widetilde{FN}(H \cup h) \cup \widetilde{TP}(h, P(H)), \end{aligned} \tag{9.8}$$

where $\widetilde{TP}(h, P(H))$ is the set of all true positives of h on $P(H)$. Moreover, by definition these two sets are non-overlapping, and thus

$$FN(H) = FN(H \cup h) \cup TP(h, P(H)). \tag{9.9}$$

For result 2, define the set

$$\widetilde{FP}(H) = \cup_{\tilde{h}\in H}\{x \in N|\tilde{h}(x) = 1\}. \tag{9.10}$$

Therefore,

$$\begin{aligned}\widetilde{FP}(H \cup h) =& \cup_{\tilde{h}\in H}\{x \in N|\tilde{h}(x) = 1\} \cup \{x \in N|h(x) = 1\} \\ =& \widetilde{FP}(H) \cup \{x \in N|h(x) = 1\}.\end{aligned} \tag{9.11}$$

By definition, $x \in N$ and $x \notin P(H)$ means that $x \in \widetilde{FP}(H)$. Thus,

$$\begin{aligned}\widetilde{FP}(H \cup h) =& \widetilde{FP}(H) \cup \{x \in N \cap P(H)|h(x) = 1\} \\ =& \widetilde{FP}(H) \cup \widetilde{FP}(h, P(H)).\end{aligned} \tag{9.12}$$

Moreover, $x \in \widetilde{FP}(H)$ means that $x \notin P(X)$, so that these two subsets do not overlap, and we thus obtain

$$FP(H \cup h) = FP(H) + FP(h, P(H)). \tag{9.13}$$

□

We can now state the primary result, which characterizes all locally optimal solutions H.

Theorem 9.5.1. *$H \subseteq \mathscr{H}$ is a local optimum if and only if either $TP(h_A, P) = 0$ or $\frac{FP(h_A,P)}{TP(h_A,P)} \geq \frac{L}{C}$.*

Proof. By definition, H is a local optimum if and only if

$$L(FN(H \cup h_A) - FN(H)) + C(FP(H \cup h_A) - FP(H)) \geq 0. \tag{9.14}$$

By Lemma 10.5.1, $FN(H \cup h_A) - FN(H) = -TP(h_A, P)$ and $FP(H \cup h_A) - FP(H) = FP(h_A, P)$,

so that a local optimum is characterized by

$$C \cdot FP(h_A, P) \geq L \cdot TP(h_A, P). \tag{9.15}$$

If $TP(h_A, P) = 0$, this inequality clearly holds. Suppose that $TP(h_A, P) \geq 1$. In that case, we see by rearranging the expression that H is a local optimum if and only if $\frac{FP(h_A,P)}{TP(h_A,P)} \geq \frac{L}{C}$. □

Below, we simplify notation by defining $FP_A \equiv FP(h_A, P)$, and defining FN_A TP_A, and TN_A similarly, with H becoming an implicit argument throughout. Now, observe that if $L/C > (1-\alpha)n$, the only locally optimal solutions have $TP_A = 0$, because otherwise $\frac{FP_A}{TP_A} \leq (1-\alpha)n < L/C$.

As a direct consequence of Theorem 10.5.1, we can bound TP_A in all locally optimal solutions.

Theorem 9.5.2. *For any locally optimal $H \subseteq \mathscr{H}$, $TP_A \leq \frac{C}{L}(1-\alpha)n$.*

Proof. If $TP_A = 0$, the result is trivially true. Suppose $TP_A \geq 1$. Then, since $\frac{FP_A}{TP_A} \geq \frac{L}{C}$, we have $TP_A = TP_A \leq \frac{C}{L} FP_A \leq \frac{C}{L} TN(H) \leq \frac{C}{L}(1-\alpha)n$. □

The upshot of Theorem 10.5.2 is that when C is small relative to L, any locally optimal H will guarantee that the attacker cannot learn a classifier that correctly identifies more than a few sensitive instances. This result also implies that an attacker with a small budget $B \leq TP_A + FP_A$ (i.e., budget is exceeded by the total number of predicted positives) can obtain very little utility from using the classifier in this case.

But what about attackers with a large budget, $B \geq TP_A + FP_A$? Clearly, when the budget is sufficiently large, the attacker will identify all the residual sensitive information in the data. However, we now show that even in this case an attacker can do little better than the trivial baseline of choosing B instances to inspect in a uniformly at random manner. An important technical consideration is that when $TP_A = 0$, an adversary can actually improve performance by prioritizing the negative predictions over the predicted positives (which

yield no utility). In this case, an adversary will likely throw away the classifier altogether. We therefore restrict our attention to the case when the attacker actually benefits from prioritizing positives over negatives. The following lemma provides a sufficient condition for this observation.

Lemma 9.5.2. *Let $B \geq TP_A + FP_A$. When $TP_A TN_A \geq FP_A FN_A$, prioritizing positive over negative instances guarantees that $U_A^* \geq U_A$ for the attacker.*

Proof. If the attacker prioritizes negatives before positives, the attacker's utility is

$$U_{A^*} = L \cdot \left(FN_A + \frac{TP_A}{TP_A + FP_A}(B - FN_A - TN_A) \right), \tag{9.16}$$

whereas the utility from the uniform random baseline is

$$U_A = L \cdot \frac{TP_A + FN_A}{TP_A + FP_A + TN_A + FN_A} B. \tag{9.17}$$

Thus, when $TP_A TN_A \geq FP_A FN_A$,

$$\begin{aligned} \frac{U_{A^*}}{U_A} &= \frac{FP_A FN_A + TP_A B - TP_A TN_A}{B} \left(\frac{TP_A + FP_A + FN_A + TN_A}{(TP_A + FN_A)(TP_A + FP_A)} \right) \\ &= \left(\frac{FP_A FN_A - TP_A TN_A}{B} + TP_A \right) \left(\frac{(TP_A + FP_A + FN_A + TN_A)}{(TP_A + FN_A)(TP_A + FP_A)} \right) \\ &\leq \left(\frac{FP_A FN_A - TP_A TN_A}{TP_A + FP_A} + TP_A \right) \left(\frac{(TP_A + FP_A + FN_A + TN_A)}{(TP_A + FN_A)(TP_A + FP_A)} \right) \\ &= 1 + \frac{(FP_A FN_A - TP_A TN_A)(FN_A + TN_A)}{(TP_A + FP_A)^2 (TP_A + FN_A)} \leq 1. \end{aligned} \tag{9.18}$$

Since U_A cannot be larger than both the utility from prioritizing positive prioritizing negative instances (being the average of these), the result follows. □

Under the condition in Lemma 10.5.2, we can now prove a bound on the the amount that the attacker can gain over the trivial baseline by using a classifier to prioritize instances, or the ratio U_A^*/U_A.

Theorem 9.5.3. *Suppose that H is a local optimum, the attacker's budget is $B \geq TP_A + FP_A$, and $TP_A TN_A \geq FP_A FN_A$. Then*

$$\frac{U_A^*}{U_A} \leq \frac{(1-\alpha)n+1}{1+\frac{L}{C}}. \tag{9.19}$$

In order to prove this theorem, we need another building block, provided by the following Lemma.

Lemma 9.5.3. *Suppose that $B \geq TP_A + FP_A$, $TP_A TN_A \geq FP_A FN_A$, and the attacker prioritizes positive instances. Then*

$$\frac{U_{A^*}}{U_A} \leq 1 + \frac{TP_A TN_A - FP_A FN_A}{(TP_A + FP_A)(TP_A + FN_A)}. \tag{9.20}$$

Proof. Suppose that the attacker prioritizes positives before negatives. Then the attacker's utility is

$$U_{A^*} = L\left(TP_A + \frac{FN_A}{FN_A + TN_A}(B - TP_A - FP_A)\right). \tag{9.21}$$

Thus,

$$\begin{aligned}
\frac{U_{A^*}}{U_A} &= \frac{TP_A TN_A + FN_A B - FP_A FN_A}{B}\left(\frac{TP_A + FP_A + FN_A + TN_A}{(TP_A + FN_A)(TN_A + FN_A)}\right) \\
&= \left(\frac{TP_A TN_A - FP_A FN_A}{B} + FN_A\right)\left(\frac{TP_A + FP_A + FN_A + TN_A}{(TP_A + FN_A)(TN_A + FN_A)}\right) \\
&\leq \left(\frac{TP_A TN_A - FP_A FN_A}{TP_A + FP_A} + FN_A\right)\left(\frac{TP_A + FP_A + FN_A + TN_A}{(TP_A + FN_A)(TN_A + FN_A)}\right) \\
&= 1 + \frac{TP_A TN_A - FP_A FN_A}{(TP_A + FP_A)(TP_A + FN_A)}.
\end{aligned} \tag{9.22}$$

□

Proof. **of Theorem 10.5.3** Since $TP_A TN_A \geq FP_A FN_A$, the attacker will prioritize positive

instances by Lemma 10.5.2. Therefore, by Lemma 10.5.3,

$$\begin{aligned}
\frac{U_{A^*}}{U_A} &\leq 1 + \frac{TP_A TN_A - FP_A FN_A}{(TP_A + FP_A)(TP_A + FN_A)} \\
&= 1 + \frac{TN_A - \frac{FP_A}{TP_A} \cdot FN_A}{\left(1 + \frac{FP_A}{TP_A}\right)(TP_A + FN_A)} \\
&\leq 1 + \frac{TN_A - \frac{L}{C} \cdot FN_A}{\left(1 + \frac{L}{C}\right)(TP_A + FN_A)} \leq 1 + \frac{(1-\alpha)n - \frac{L}{C}}{1 + \frac{L}{C}} \\
&= \frac{(1-\alpha)n + 1}{1 + \frac{L}{C}}.
\end{aligned} \tag{9.23}$$

□

The upshot of Theorem 10.5.3 is that even when an attacker with a large budget cannot do much better than uniformly selecting instances to inspect. As an example, consider again Figure 10.1, which illustrates the result after the application of the set of classifiers H. It can be seen that there are 26 instances in total, with a breakdown of 3 *true positives*, 6 *false positives*, 15 *true negatives*, and 2 *false negatives*. Now, if the attacker has a budget of $B = 20$, $\frac{U_{A^*}}{U_A} = \frac{3 + (20-3-6)\frac{2}{2+15}}{20\frac{3+2}{26}} \approx 1.11$.

9.5.2 Finite Sample Bounds

Armed with the idealized generic analysis of locally optimal classifier subsets H, we can generalize these results to account for finite sampling error. While the results in the previous section are applicable for arbitrary locally optimal subsets, our finite sample analysis is specific to *GreedySanitize*.

Consider the point at which the publisher halts the greedy data sanitization Algorithm 16 and publishes the data (after applying the resulting set of classifiers H to it). If only a few training data points remain, the publisher's decision would entail significant risk because the error in estimating the relevant decision parameters will be quite high. As such, in this case, no data should be published. We therefore consider the case when there

is a non-trivial amount of training data remaining after Algorithm 16 terminates. As our experiments below demonstrate, this is a reasonable assumption to invoke in practice. In the following discussion, we denote the size of this residual training data m.[2]

Our point of departure is the standard learning-theoretic framework. To simplify the presentation, we assume that the published data set is sufficiently large, so that the relevant quantities (e.g., the number of true positives) are close to their expected values on randomly chosen data sets of the same size. Now, let our hypothesis class $\mathscr{H}$ contain a set of functions from a set X to $\{0,1\}$, and assume $\mathscr{H}$ has finite Vapnik-Chervonenkis dimension $v \geq 1$. Suppose that P is the data set remaining after Algorithm 16 terminates and the resulting classifiers H are applied to the original data X. Let the classifier used in the last iteration by Algorithm 16 be $\widehat{h_A}$, which is only optimal on training data. In other words, $\widehat{h_A}$ is the publisher's approximation of the classifier h_A that would subsequently be applied by the attacker to P. Let $\widehat{FN_A}$, $\widehat{FP_A}$, $\widehat{TP_A}$, $\widehat{TN_A}$ be the corresponding approximate counts of false negatives, false positives, etc., applying $\widehat{h_A}$ to the training data, whereas FN_A, FP_A, TP_A, and TN_A still denote the corresponding counts for the actual optimal classifier h_A that the attacker would use. The attacker's corresponding utility, estimated using the training data, is denoted by $\widehat{U_A^*}$, while the actual attacker utility is U_A^*. The utility for the attacker gained from the baseline policy is still U_A.

We start by noting the well-known error bound connecting empirical and actual errors in classification:

$$\frac{\widehat{FP_A} + \widehat{FN_A}}{m} \leq \frac{FP_A + FN_A}{m} + \lambda(\delta, m) \tag{9.24}$$

with probability at least $1 - \delta$, where

$$\lambda(\delta, m) = \left(\frac{41}{m} \left(v \log\left(\frac{2em}{v} \right) + \log\left(\frac{4}{\delta} \right) \right) \right)^{\frac{1}{2}}.$$

[2]For simplicity, we assume that m is also the size of the residual test data that is ultimately released. Generalization of the results below is relatively direct.

For our purposes, however, this result is not sufficient. For example, there may be two classifiers, h and h' in $\mathscr{H}$ with a similar error, but with very different numbers of false positives and false negatives. Thus, in order to bound the utility of the attacker, we need to call upon several additional assumptions. Specifically, we make the following assumptions: $\widehat{FP_A} \leq p\widehat{FN_A}$, $\widehat{TP_A} \geq q\widehat{N_A}$, $FP_A \geq sFN_A$, and $TP_A \leq rN_A$. Since the parameters p,q,s,r can be arbitrary, these relationships are quite general. However, the results below are most meaningful if these bounds are tight.

Lemma 9.5.4. *Suppose that* $\widehat{TP_A} \geq 1$ *when Algorithm 16 terminates. Then,*

$$\frac{FP_A}{TP_A} \geq \left(\frac{1}{1+\frac{1}{s}}\right)\left((1+\frac{1}{p})\cdot q\cdot\frac{L}{C}-\lambda(m,\delta)\right)\frac{1}{r}$$

with probability at least $1-\delta$.

Proof. By Theorem 10.5.1, Algorithm 16 will terminate when $\frac{\widehat{FP_A}}{\widehat{TP_A}} \geq \frac{L}{C}$. Using, Equation 10.24 and our assumptions, we have

$$\frac{(1+\frac{1}{p})\widehat{FP_A}}{m} \leq \frac{(1+\frac{1}{s})FP_A}{m}+\lambda(m,\delta)$$

with probability at least $1-\delta$. Consequently, with probability at least $1-\delta$,

$$\frac{(1+\frac{1}{p})\widehat{FP_A}}{TP_A\cdot\widehat{TP_A}} \leq \frac{(1+\frac{1}{s})FP_A}{TP_A\widehat{TP_A}}+\lambda(m,\delta)\frac{m}{\widehat{TP_A}TP_A},$$

and, consequently,

$$\frac{1+\frac{1}{p}}{TP_A}\cdot\frac{L}{C} \leq \frac{(1+\frac{1}{s})FP_A}{TP_A\widehat{TP_A}}+\lambda(m,\delta)\frac{m}{\widehat{TP_A}TP_A}.$$

Rearranging, we get

$$\begin{aligned}
\frac{FP_A}{TP_A} &\geq \frac{\left(1+\frac{1}{p}\right)\frac{L}{C}\cdot\frac{1}{TP_A} - \lambda(m,\delta)\cdot\frac{m}{\widehat{TP_A}}\cdot\frac{1}{TP_A}}{\left(1+\frac{1}{s}\right)\cdots\frac{1}{\widehat{TP_A}}} \\
&= \frac{1}{1+\frac{1}{s}}\left(\frac{L}{C}\left(1+\frac{1}{p}\right)\cdot\frac{\widehat{TP_A}}{m}\cdot\frac{m}{TP_A} - \lambda(m,\delta)\cdot\frac{N_A}{TP_A}\right) \\
&\geq \left(\frac{1}{1+\frac{1}{s}}\right)\left(\left(1+\frac{1}{p}\right)\frac{\widehat{TP_A}}{m}\frac{L}{C} - \lambda(m,\delta)\right)\frac{m}{TP_A} \\
&\geq \left(\frac{1}{1+\frac{1}{s}}\right)\left(\left(1+\frac{1}{p}\right)\cdot q\cdot\frac{L}{C} - \lambda(m,\delta)\right)\frac{1}{r}
\end{aligned}$$

□

Clearly, the bound in Lemma 10.5.4 is only meaningful when $\lambda(m,\delta) \leq (1+\frac{1}{p})q\frac{L}{C}$, that is, for a sufficiently large sample m. Therefore, the results below assume this to be the case.

Building on the result in Lemma 10.5.4, we can now extend the bounds on the attacker's success developed in Section 10.5.1 to account for finite sample error.

Theorem 9.5.4. *When Algorithm 16 terminates,*

$$TP_A \leq r\left(1+\frac{1}{s}\right)\frac{(1-\alpha)n}{(1+\frac{1}{p})\cdot q\cdot\frac{L}{C} - \lambda(m,\delta)}$$

with probability at least $1-\delta$.

Proof. Since

$$\begin{aligned}
\frac{FP_A}{TP_A} &= \frac{TN_D - TN_A}{TP_A} \\
&\geq \left(\frac{1}{1+\frac{1}{s}}\right)\left((1+\frac{1}{p})\cdot q\cdot\frac{L}{C} - \lambda(m,\delta)\right)\frac{1}{r}
\end{aligned}$$

$$
\begin{aligned}
TP_A &\le r\left(1+\frac{1}{s}\right)\left(\frac{1}{(1+\frac{1}{p})\cdot q\cdot\frac{L}{C}-\lambda(m,\delta)}\right)(TN_D-TN_A)\\
&\le r\left(1+\frac{1}{s}\right)\left(\frac{1}{(1+\frac{1}{p})\cdot q\cdot\frac{L}{C}-\lambda(m,\delta)}\right)\cdot TN_0\\
&= r\left(1+\frac{1}{s}\right)\frac{(1-\alpha)n}{(1+\frac{1}{p})\cdot q\cdot\frac{L}{C}-\lambda(m,\delta)}.
\end{aligned}
$$

□

Theorem 9.5.5. *Suppose that $TP_ATN_A \ge FP_AFN_A$, and $B \ge TP_A + FP_A$. Then,*

$$
\frac{U_A^*}{U_A} \le \frac{((1-\alpha)n+1)\,r(1+\frac{1}{s})}{r(1+\frac{1}{s})+(1+\frac{1}{p})q\frac{L}{C}-\lambda(\delta,m)}
$$

with probability at least $1-\delta$.

Proof.

$$
U_A = L\cdot B\cdot\frac{TP_A+FN_A}{TP_A+FP_A+FN_A+TN_A}.
$$

Based on the general adversarial model, the attacker can always choose the priority to guarantee $U_{A^*} \ge U_A$ according to Lemma 10.5.2. Therefore, when $TP_ATN_A \ge FP_AFN_A$, the attacker prioritizes the positives than negatives, so $U_{A^*} = L\cdot\left(TP_A+\frac{FN_A}{FN_A+TN_A}(B-TP_A-FP_A)\right)$.

Therefore we have

$$\begin{aligned}
\frac{U_{A^*}}{U_A} &= 1 + \frac{TP_A TN_A - FP_A FN_A}{(TP_A + FP_A)(TP_A + FN_A)} \\
&= 1 + \frac{TN_A - \frac{FP_A}{TP_A} \cdot FN_A}{\left(1 + \frac{FP_A}{TP_A}\right)(TP_A + FN_A)} \\
&\leq 1 + \frac{TN_A - \frac{FP_A}{TP_A}}{1 + \frac{FP_A}{TP_A}} \\
&\leq 1 + \frac{(1-\alpha)nr(1+\frac{1}{s}) - ((1+\frac{1}{p})q\frac{L}{C} - \lambda(\delta, m))}{r(1+\frac{1}{s}) + (1+\frac{1}{p})q\frac{L}{C} - \lambda(\delta, m)} \\
&= \frac{((1-\alpha)n+1)\, r(1+\frac{1}{s})}{r(1+\frac{1}{s}) + (1+\frac{1}{p})q\frac{L}{C} - \lambda(\delta, m)}.
\end{aligned}$$

□

9.6 Experiments

In this section, we assess the performance of *GreedySanitize* (GS) using two electronic health record data sets to protect the personal sensitive identifiers (here we only consider the individuals' names): 1) publicly accessible medical records from the I2B2 corpus [211] and 2) a private electronic medical records (EMR) dataset from the Vanderbilt University Medical Center (VUMC). In addition, we evaluate the performance of our model on two non-health-related data sets to assess its generalizability: 1) Enron email data and 2) newsgroup data [235]. In both of these, we also treat individuals' names as sensitive entities. The following statistics provide some intuition into the size and complexity of these resources:

- i2b2: contains 386,736 words in 664 documents. 6853 words are labeled as sensitive entities and involve synthetic names in place of actual patient identifiers.
- VUMC: contains 226,455 words in 600 documents, with 5154 labeled as sensitive. Unlike the i2b2 corpus, these entities correspond to real patient identifiers.

- Enron: contains 120,131 words in 761 emails, with 6084 names which we labeled as sensitive.
- Newsgroup: contains 119,303 words in 597 documents, of which 3525 (names) are sensitive.

We used three state-of-the-art learning algorithms for sensitive entity recognition: conditional random fields (CRF), which consistently ranks as the best method for identifying personal health information in electronic medical records [211, 207, 212], support vector machine (SVM) [236], which makes use of the features of the word itself, part-of-speech (POS), morphologic information, and the history class of previous features assigned by the classifier; as well as a recently proposed ensemble method [62], which applies CRF to classify first and then uses SVM to reduce the false positives. All these play a dual-role in our experiments: they serve as a comparison baseline to prior art, as well as the core learning algorithms in our own Algorithm 16 (GreedySanitize). In all the experiments, the attacker first runs all three of these algorithms on the training holdout from published data, and then chooses the best performing classifier. Our evaluation is based on four-fold cross-validation, with GreedySanitize running on the training data and using the incidence of true and false negatives on training data to determine when to stop.

9.6.1 Privacy Risk

When the budget of the attacker is small, our theoretical results provide an upper bound on the expected number of identified instances. While this upper bound suggests that risk becomes arbitrarily small when the associated loss is large, it is not tight. In Figure 10.4 we demonstrate that the number of identified instances (which is equivalent to the number of true positives for the attacker's classifier) typically becomes negligible even when L is quite small relative to C. An interesting exception is the VUMC dataset, where the number of identified instances remains relatively large until the loss from re-identification is quite

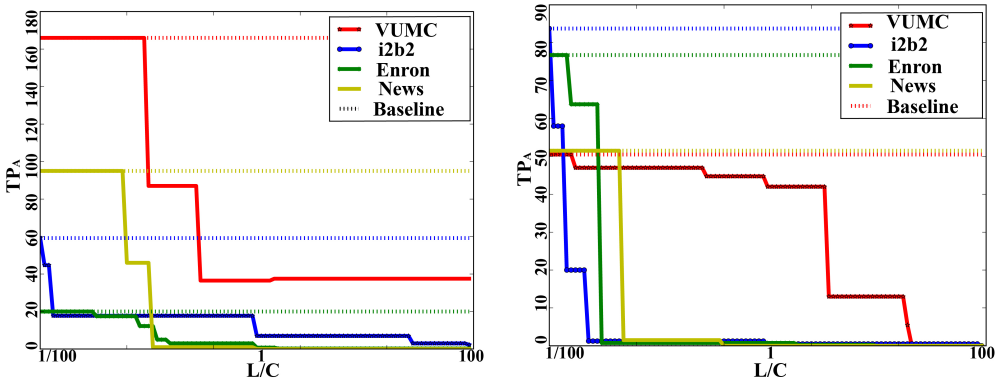


Figure 9.4: The number of residual *true positive* instances TP_A (equivalently, identified instances for an attacker with a small budget) after running GreedySanitize for the i2b2, VUMC, Enron, and Newsgroup datasets. (a) GreedySanitize using CRF (dashed lines, or baseline, correspond to standard application of CRF). (b) GreedySanitize using the best classifier from {CRF, SVM, Ensemble} (dashed lines correspond to the baseline application of the best classifier from this collection).

high.

To investigate privacy risk more generally, we now consider the expected number of identified instances as a function of adversary's budget (and normalized by the budget). To make a meaningful comparison to the state of the art classification schemes, we apply them in a cost sensitive manner, so that L becomes the cost of false negatives and C the cost of false positives, just as in our model. Figure 10.5 compares the proposed GS algorithm to the cost sensitive state-of-the-art CRF, SVM, and Ensemble algorithms using the same values of L and C in GS and cost sensitive versions of the classifiers. We can see that for the same values of L/C, the proposed GS algorithm is consistently competitive with, or better than the best state-of-the-art cost sensitive alternatives in terms of privacy risk, except when adversary's budget is extremely low. However, with low budget, privacy risk is negligible for sufficiently high L/C (Figure 10.4).

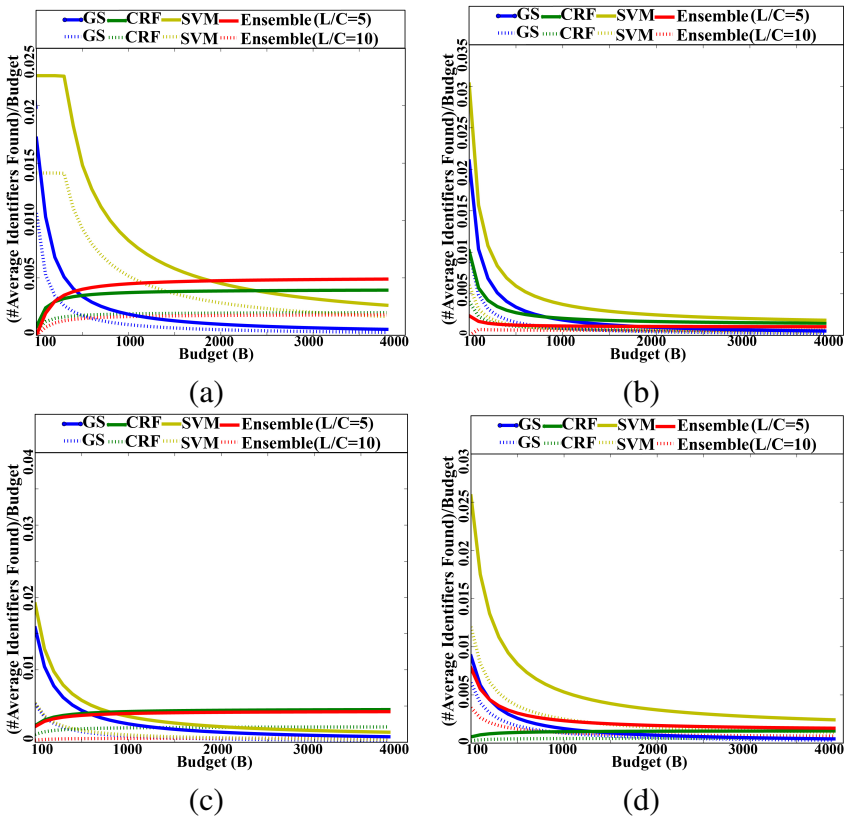


Figure 9.5: The ratio of average sensitive identifiers found by the attacker and the adversarial budget, while the publisher applies different classifiers with cost sensitive learning as $L/C = 5, 10$. (a)-(d) corresponds to the i2b2, VUMC, News, and Enron datasets.

9.6.2 Data Utility

Our next evaluation concerns whether we can still retain the data utility with such a high privacy preserving requirement. This is something that motivates the presented approach (as compared to simply suppressing all data), but that we did not explicitly consider in the theoretical analysis. Intuitively, the proposed *GreedySanitize* algorithm should strike a reasonable balance: it stops immediately after a local optimum is reached. In our model, of course, there may be multiple local optima thereafter, but these would result in less data being published. Here, we evaluate the data utility of the published data using the *publish ratio*, which is defined as the proportion of the original number of entities in the published data.

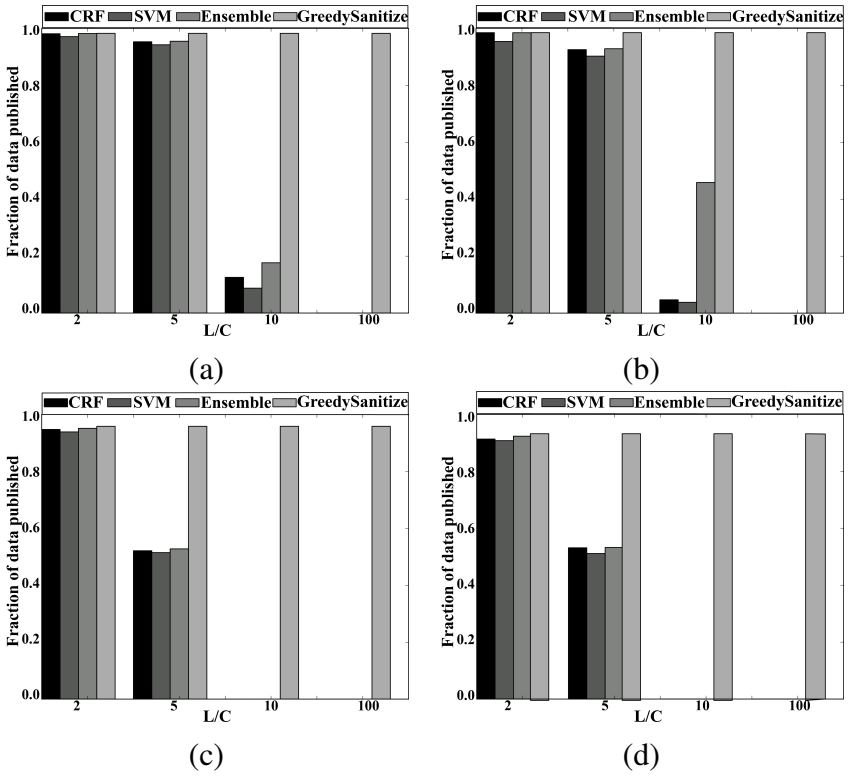


Figure 9.6: Fraction of data published for different classifiers with cost sensitive learning. (a)-(d) corresponds to the i2b2, VUMC, News, and Enron datasets.

Figure 10.6 compares GreedySanitize to cost-sensitive variants of the baseline algorithms (CRF, SVM, and Ensemble). GreedySanitize preserves most of the data utility even when L/C is high. Specifically, in both of the EMR datasets over 98% of the data is published, *even when L/C is quite high*. The performance for the other two data sets is lower, but still, over 93% of the data is ultimately published, even with large L/C ratios. In contrast, when the loss due to re-identification is moderate or high, cost-sensitive algorithms essentially suppress most of the data, resulting in very low utility. GreedySanitize therefore offers a far better balance between risk and utility than the state-of-the-art alternatives.

9.6.3 Impact of the Size of the Hypothesis Space

One important issue in applying GreedySanitize is that perhaps the attacker will make use of a new algorithm that the publisher had not considered. We now explore this issue by

considering the quality of decisions when the publisher uses a single classifier, or the best of all three that we consider, at the core of GS.

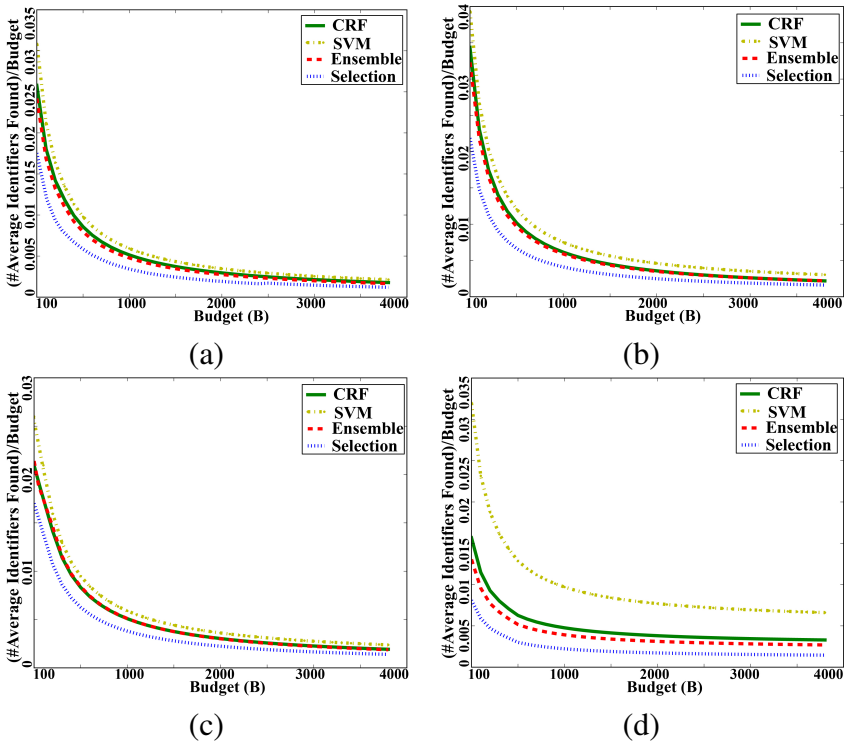


Figure 9.7: The ratio of average sensitive identifiers found by the attacker and the adversarial budget, while the publisher applies classifiers CRF, SVM, Ensemble, and Selection which allows the publisher to choose a learner with highest accuracy from {CRF, SVM, Ensemble} for GreedySanitize (L/C=5). (a)-(d) corresponds to the i2b2, VUMC, News, and Enron datasets.

Figures 10.7 and 10.8 compare these four options (the three single-classifier options, and the last, called "Selection", where the most accurate of these three classifiers is chosen in each iteration), evaluated when the adversary chooses the most accurate of these. The overall observation is that while increasing the space of classifiers to choose from does help, the difference is relatively small, so that significant underestimation of the attacker's strength appears unlikely to make much impact.

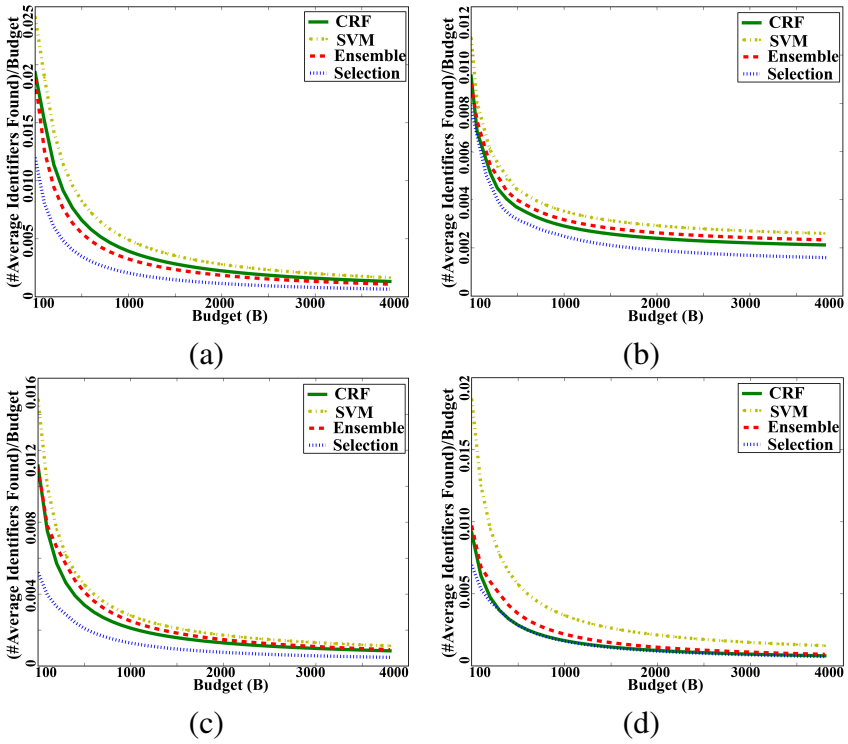


Figure 9.8: The ratio of average sensitive identifiers found by the attacker and the adversarial budget, while the publisher applies classifiers CRF, SVM, Ensemble, and Selection which allows the publisher to choose a learner with highest accuracy from {CRF, SVM, Ensemble} for GreedySanitize (L/C=10). (a)-(d) corresponds to the i2b2, VUMC, News, and Enron datasets.

9.6.4 Number of Greedy Iterations

The final issue we consider is the number of iterations of GreedySanitize (and, consequently, the number of classifiers it uses) for the different data sets. Figure 10.9 shows that for all four datasets (and for the entire range of L/C that we consider) the average number of iterations is less than 5. Our theoretical upper bound is, therefore, extremely pessimistic. Indeed, for some datasets, such as the VUMC EMR dataset, the average number of iterations is just above 2 even when the loss from leaking sensitive information is quite high.

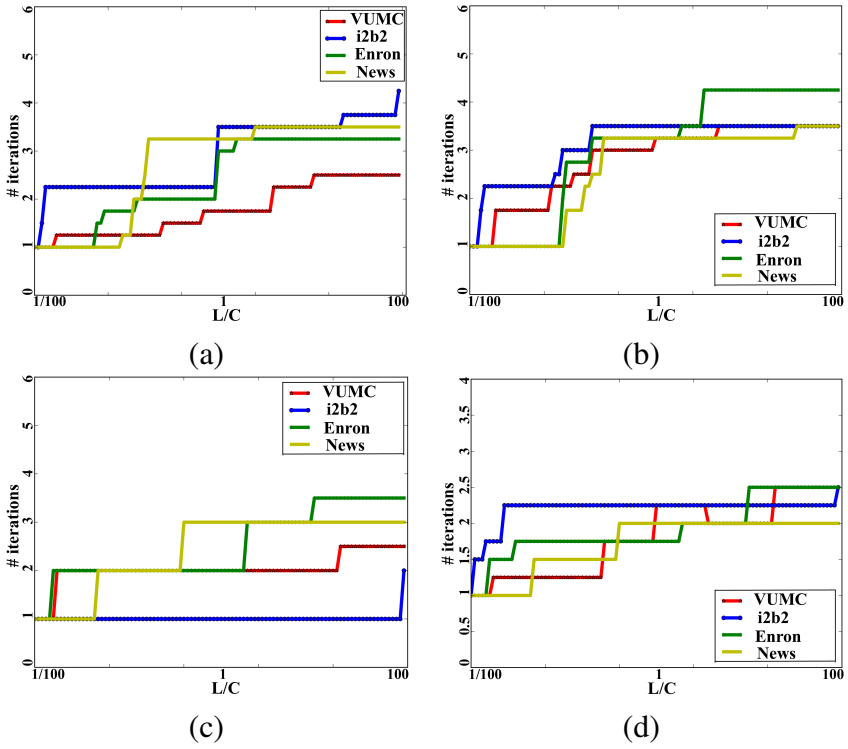


Figure 9.9: The number of iterations of GreedySanitize for i2b2, VUMC, Enron, and Newsgroup datasets (a)-(d) respectively, where publisher chooses CRF, SVM, Ensemble, and the best algorithm from {CRF, SVM, Ensemble}, respectively.

9.7 Summary of Contributions

Based on the PPDP data sharing requirement for knowledge-based decision making and data mining, how to guarantee the privacy preserving purpose as well as retain the data utility has become a challenge during the data publishing process. If the data publisher is too conservative, it will largely reduce the published data utility; if the data publisher loosens the de-identification boundary, it would lead to high privacy leaking. Therefore, we form the aggregate classifier framework to allow the data publisher iteratively find and de-identify the sensitive identifiers based on his preference for the tradeoff between privacy preserving and data utility.

Overall, the contributions are: 1) Provide the aggregate framework to perform de-identification and theoretically prove that the attackers cannot gain more utility than that

they uniformly randomly choose instances and its surrounding context to inspect, even they trained an efficient classifier to help rank the instance inspection priority based on the perfect knowledge of the data publisher's learning algorithms; 2) Propose the greedy algorithm for selecting the near optimal sequence of classifiers for defender considering both of the privacy preserving purpose and the data utility; 3) Provide the sample complexity analysis for the greedy algorithm and essentially offer the robust algorithm on the unknown distribution of dataset; 4) Apply the proposed algorithm to various large datasets for evaluation, therefore demonstrate not only the robustness but also the scalability of the proposed algorithm.

Chapter 10

CONCLUSION AND FUTURE WORK

10.1 Contributions

The adversarial machine learning in security-sensitive domains is an important and rapidly expanding sub-discipline that integrates machine learning, computer security, and game theory. In this thesis, I aim to provide *secure learning* by analyzing the properties and limitation for different attacks, as well as designing robust learning algorithms against the real world sophisticated adversarial strategies.

To avoid security pitfalls, reasonable threat models must be developed first for potential adversaries and then accordingly design learning systems to meet the desired security requirements. We explore the properties and possibilities of evasion attacks in Chapter 4 and the poisoning attacks in Chapter 9. The constraints of adversaries can come from the knowledge about the learning algorithm itself, the number of training data that allowed to be injected, and the cost of modifying data points for adversaries, etc. The important limitation for learning systems is how to balance the trade-offs between a learner's performance on regular data and its resilience to attacks. Understanding these trade-offs is crucial not only for security applications but also for understanding how learners behave in any non-ideal settings. Therefore in this thesis I designed robust learning algorithms targeting on specific kinds of attacks, as well as general robust defensive framework which can take any learning algorithm and attack models in and output the robust learning results. Additionally, we also take care of the scalability issue in Chapter 8 to process massive amounts of data available for Internet-scale problems to realize the goal of secure learning for large scale dataset. Besides, we have also considered the the data privacy issues within this thesis in adversarial environments to make current privacy preserving learning system more

robust.

10.2 Future Work

Game theoretic models and analyses have demonstrated the ability to make intelligent decisions for security challenges faced by learning systems. Therefore I plan to continuously focus on both the theoretical analysis of more general threat models, and developing practical systems to improve the system robustness against not only evasion but also poisoning attacks. I will also evaluate the uncertainty about threat models and develop more realistic systems to preserve robustness, as well as optimize the resource allocation based on real-world constraints, such as the fault tolerance level. In addition to expanding the expressiveness and usability of the robust adversarial learning approach, I am also interested in generalizing the approach to other concerns, in particular domain-specific concerns in social applications. All these techniques would be developed and integrated with cloud computing infrastructure as one of my targets in order to realize the goal of *secure learning* for big data.

10.2.1 Robust crowd sourcing mechanism design

Crowdsourcing has gained immense popularity. In machine learning applications, for example, human workers are paid over crowdsourcing platforms such as Amazon Mechanical Turk (AMT) for labeling data. In doing so, machine learning researchers can easily obtain the ground truth for large datasets. Despite its flexibility and scalability, crowdsourcing approaches main challenge is guaranteeing the quality of the collected data. Currently, workers can get the same amount of payment even if they provide wrong labels either intentionally or unintentionally. This problem will be amplified when collecting data for security applications such as spam detection where adversarial workers might intentionally poison the dataset. To address this challenge, we plan to design adaptive payment mechanisms to incentivize workers to either answer correctly or point out other malicious/wrong answer-

s/labels. To achieve this goal, we will model the interactions among different workers as a game. By analyzing the equilibria of this game, we aim to induce lower error rates under the proposed mechanism while introducing as small overhead on monetary expenditure as possible. We also plan to conduct empirical evaluations based on AMT to validate the design.

10.2.2 Secure classification against poisoning attacks based on robust matrix completion

Current robust learning algorithm implementations are designed for evasion attacks, where adversaries try to manipulate their malicious inputs and evade detection. However, from cheap ubiquitous computing, crowd-sourcing and data flow collection are employed nowadays to do real time analysis for large scale collected datasets. Therefore, poisoning attacks are easy to conduct and attackers have invested a large amount of effort to compromise weak endpoints, such as sensors or network nodes to send fake information to mislead the learning system. So I plan to study the poisoning attack to develop a robust learning framework against poisoning attacks. I plan to consider the feature matrix as a combination of a low rank matrix and a sparse matrix with arbitrary errors. Therefore we can apply the robust matrix completion technique to resume the most "benign" entries of the feature matrix. Then train a classifier based on this recovered matrix to derive the error bound for the final classification result. The advantage of this is that we do not need to make any distribution assumption for the normal data and can tolerant arbitrarily large corruptions, while current robust regression or classification algorithm all need to assume the normal data is from sub-Gaussian distribution. I acquired a real-world dataset of web proxy logs from RSA lab to classify the malicious domains against poisoning attacks by evaluating the feature correlation before and after poisoned data are injected. I will first prove the optimal injection strategy of the attacker, then develop the robust detection algorithm to detect as well as remove the poisoning data points and retrain the system to enhance its robustness.

10.2.3 Adversarial-aware learning in social networks

First I plan to enhance the privacy protection within social networks. There are several work for differential privacy on recommendation systems using different algorithms for matrix completion by adding gaussian noise. Joint differential privacy notion is also introduced and it would be interesting to explore how to make different state-of-art matrix completion algorithms differentially private. Currently, only the SGD method is made differentially private, but there is no any theoretic guarantee since it is based on the SGLD. So I plan to make ALS or other algorithms differentially private for recommendation systems, and derive theoretic bounds for the utility, privacy and convergence since ALS and other matrix completion algorithms themselves have good bound already.

Besides the privacy issue, social network can become a target for adversaries given the fact that it can help propagate the influence through social medias. However, all current models, including the linear threshold, independent cascade and decreasing evaluated, do not take into account presence of an adversary in the social network. The attacker can have malicious goals, such as to prevent diffusion of ideas that an opponent is trying to promote; promote certain ideas to maximize his benefits; randomly perturb the diffusion to decrease the reliability of the current network to further explore his malicious strategies. For example, twitter may want to decrease the reliability of face book and gain more users for twitter. Therefore, I plan to 1). Anomaly detection based the node or sub-network connections; 2). Robustly recover the most trustful links based on external information instead of simply the flow of each connection, since the adversary can manipulate the flow for certain links.

Additionally, I also plan to apply Gibbs sampling to network structures to optimize the network efficiency based on the designed cost function Based on a set of network structures or we can even use deep learning to generate some network structures based on the existing ones, we can run a Gibbs sampling to select the most suitable network for certain diffusion purpose. This could be a general framework for designing efficient network structures. Or,

given the existing nodes, we can sample the connections to make the network more efficient in terms of the cost functions we designed, which can be made a personalized system.

10.2.4 Deep learning in adversarial environments

Current work for "adversarial examples" all generate adversarial noise for deterministic deep networks. We have found that by adding random noise during learning, the deep network will be more robust against sophisticated adversaries. However, it is unclear to what extend such non-deterministic deep networks are robust; and whether there are adversarial examples that could be generated for such non-deterministic learning process. I plan to explore the adversarial ability to compromise the non-deterministic deep networks or without full knowledge of the network structures. Therefore, more general attacks and corresponding defensive strategies would be derived to meet the real world cases where the deep network structure is unavailable.

Appendix A

Appendix for chapter 5

A.1 Supplemental comparison results for 500 features

Here we test the AAS scheme with the same set up of simulations on the feature space of 500, and similar results shown as below have demonstrated the consistency and robustness of our proposed approach.

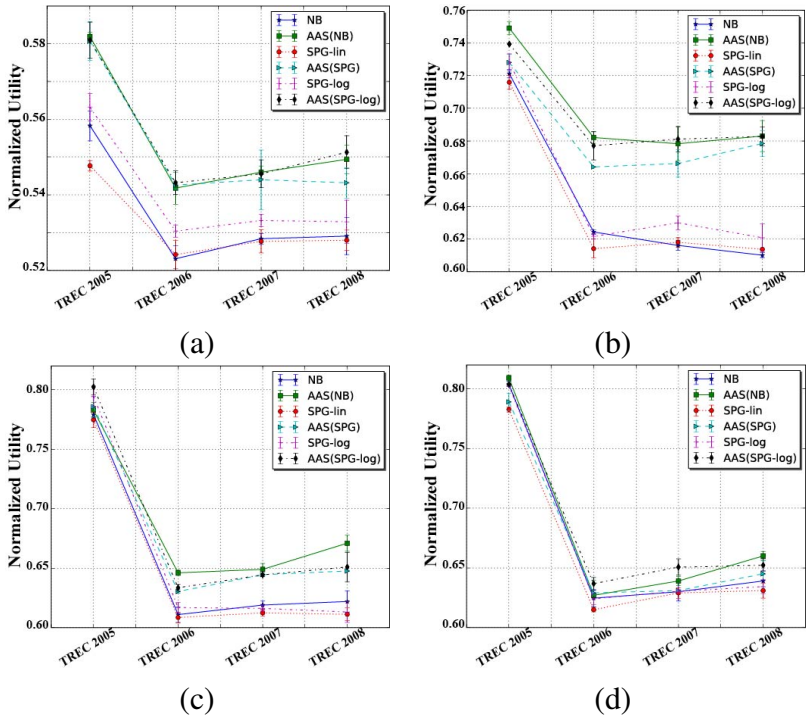
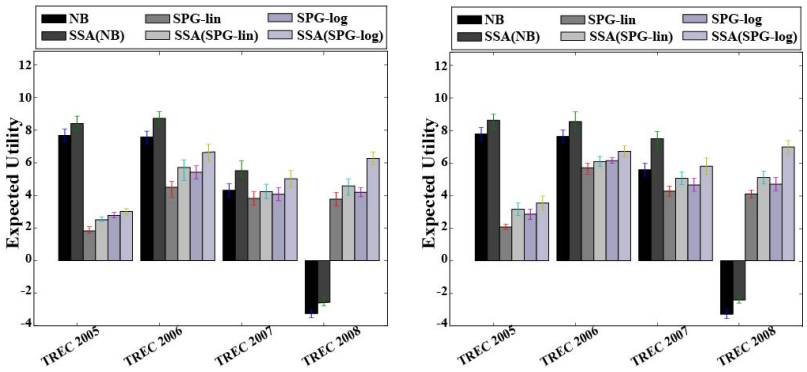
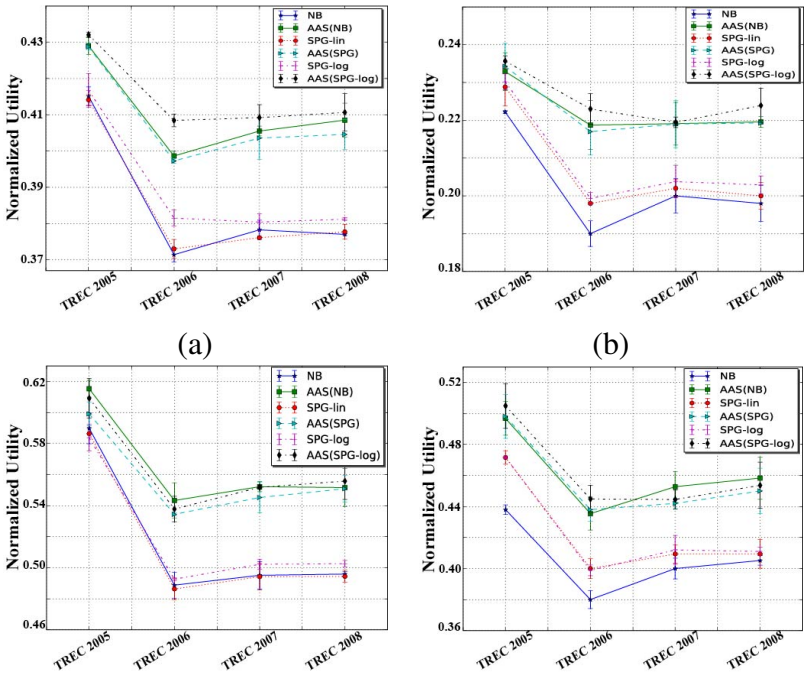


Figure A.1: Comparison of normalized utility on TREC data, trained on year 2005, and tested on years 2005-2008. Our method is labeled as $AAS(\cdot)$, where the parameter is the classifier that serves to provide $p(x)$. The following parameters are used: $\delta = 0.2$,$V(x) = G(x) = 1$, $P_A = 1$ (a) c=0.1; (b) c=0.3; (c) c=0.5; (d) c=0.9.



(a) (b)

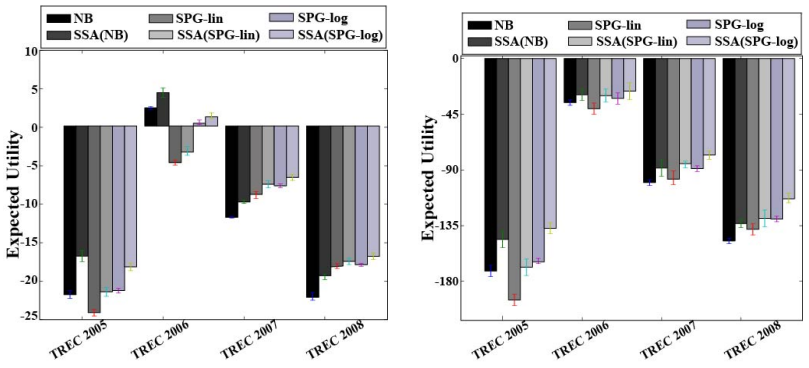
Figure A.2: Comparison of the expected utility assuming $P_A = 1$, $V(x) = G(x) = 1$; (a) $c = 0.1$; (b) $c = 0.3$.



(a) (b)

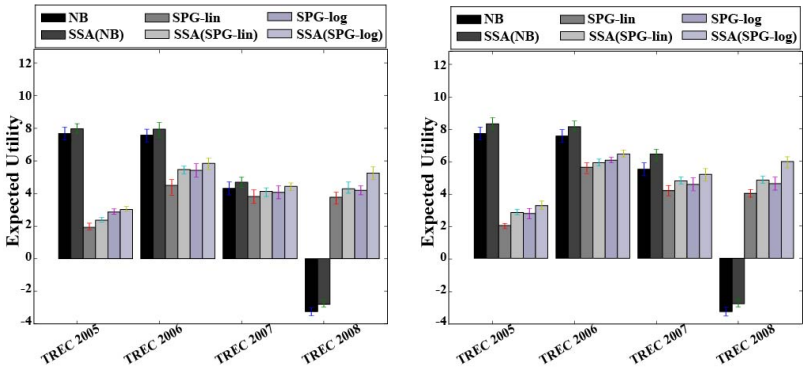
(c) (d)

Figure A.3: Comparison of normalized utility on TREC data, trained on year 2005, and tested on years 2005-2008. Our method is labeled as $AAS(\cdot)$, where the parameter is the classifier that serves to provide $p(x)$. The following parameters are used: $\delta = 0.2$,$G(x) = 1$, $P_A = 1$ (a) $V(x) = 2$, c=0.1; (b) $V(x) = 10$, c=0.1; (c) $V(x) = 2$, c=0.3; (d) $V(x) = 10$, c=0.3.



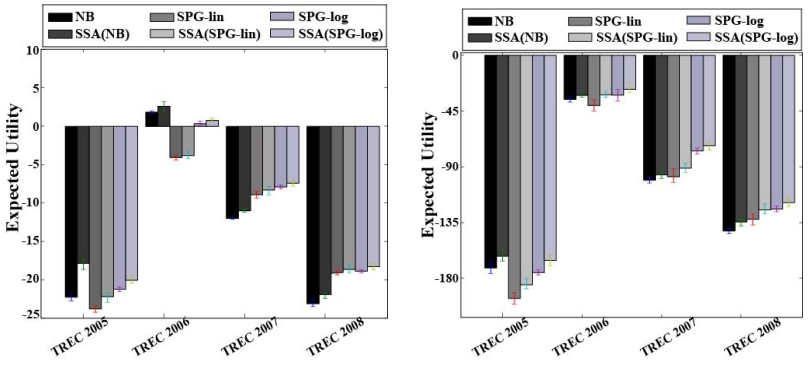
(a) (b)

Figure A.4: Comparison of the expected utility assuming $P_A = 1$; (a) $V(x) = 2$; (b) $V(x) = 10$. $c = 0.3$.



(a) (b)

Figure A.5: Comparision of the expected utility assuming $P_A = 1$, introducing parameter error with 0.2 for δ; (a) $c = 0.1$; (b) $c = 0.3$.



(a) (b)

Figure A.6: Comparison of the expected utility assuming $P_A = 1$, introducing parameter error with 0.2 for δ; (a) $V(x) = 2$; (b) $V(x) = 10$. $c = 0.3$.

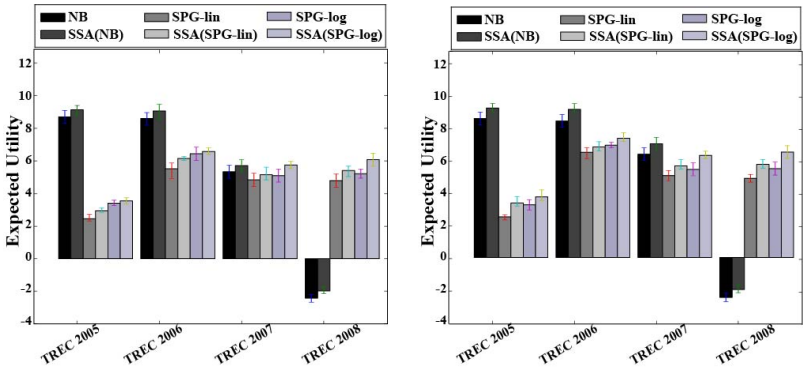


Figure A.7: Comparison of the expected utility assuming $P_A = 1$, introducing adversarial model error; (a) $c = 0.1$; (b) $c = 0.3$.

Appendix B

Appendix for chapter 6

B.1 User Interface Details

B.1.1 English Test

We screened participants to ensure English language proficiency by using the online English test `http://www.easyenglish.com/index.asp`. We took three questions from this test, shown in Figure B.1. To pass, the subjects had to answer two of these correctly.

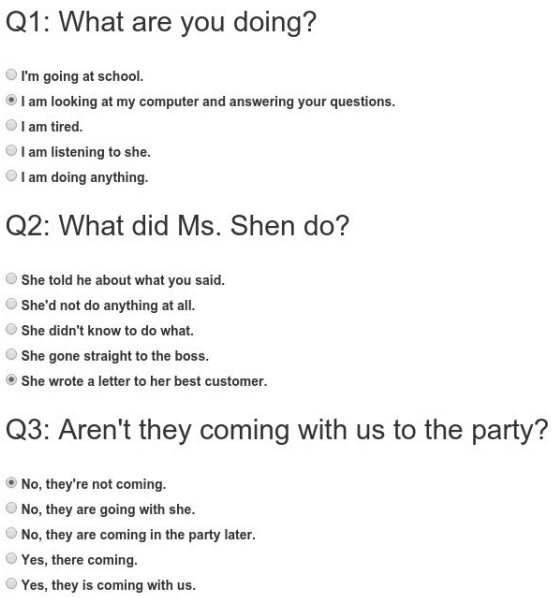
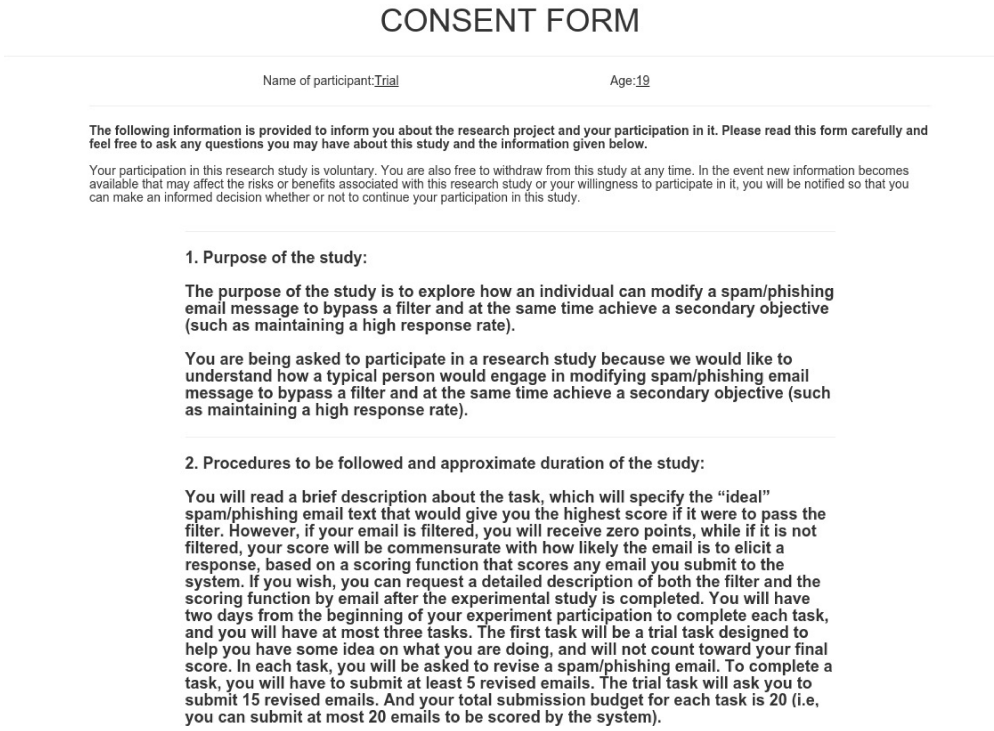


Figure B.1: English language test questions.

B.1.2 Consent Form

Prior to starting the experimental tasks, the subjects were asked to read a consent form (Figure B.2) and indicate agreement to participate, as well as indicate that they were at least

18 years old, but clicking the “Agree” button.



CONSENT FORM

Name of participant:Trial Age:19

The following information is provided to inform you about the research project and your participation in it. Please read this form carefully and feel free to ask any questions you may have about this study and the information given below.

Your participation in this research study is voluntary. You are also free to withdraw from this study at any time. In the event new information becomes available that may affect the risks or benefits associated with this research study or your willingness to participate in it, you will be notified so that you can make an informed decision whether or not to continue your participation in this study.

1. Purpose of the study:

The purpose of the study is to explore how an individual can modify a spam/phishing email message to bypass a filter and at the same time achieve a secondary objective (such as maintaining a high response rate).

You are being asked to participate in a research study because we would like to understand how a typical person would engage in modifying spam/phishing email message to bypass a filter and at the same time achieve a secondary objective (such as maintaining a high response rate).

2. Procedures to be followed and approximate duration of the study:

You will read a brief description about the task, which will specify the “ideal” spam/phishing email text that would give you the highest score if it were to pass the filter. However, if your email is filtered, you will receive zero points, while if it is not filtered, your score will be commensurate with how likely the email is to elicit a response, based on a scoring function that scores any email you submit to the system. If you wish, you can request a detailed description of both the filter and the scoring function by email after the experimental study is completed. You will have two days from the beginning of your experiment participation to complete each task, and you will have at most three tasks. The first task will be a trial task designed to help you have some idea on what you are doing, and will not count toward your final score. In each task, you will be asked to revise a spam/phishing email. To complete a task, you will have to submit at least 5 revised emails. The trial task will ask you to submit 15 revised emails. And your total submission budget for each task is 20 (i.e, you can submit at most 20 emails to be scored by the system).

Figure B.2: The online consent form.

B.2 “Ideal” Email Templates

Each task involved a random assignment of one of 10 “ideal” email templates which the subjects needed to subsequently modify (each of these was filtered by our classification-based filter). In this section we present the text of all of these email instances (all are actual spam/phishing emails). 4 of the 10 instances were spam emails, and the remaining 6 were phishing emails.

B.2.1 Instance 1 (Spam)

Save 80% discount on drugs . . . save 80% on every order !

We are the number one online retailler for dozens of medications. Our customers save 80 cents out of every dollar, every time, compared to the industry price. Yes, that is less

than quarter - price

We have all the products that our customers have asked for, including new superviagra soft-tabs that work in just 15 minutes ! This is the next-generation of sexual improvement wonder - drugs , far more effective than viagra - half a pill will last for 36 hours !

Get all the information on superviagra here : ***malicious url***

Our keys to keeping customers satisfied are:

Easy ordering online Save 80% on regular price

We have massive stocks of drugs for same day dispatch Fast delivery straight to your door with discrete packaging We are the biggest internet retailler with thousands of regular customers

No consultation fee

No intimate questions or examinations

No appointment

No prior prescription needed

private and confidential service

Please come by our shop, see for yourself the massive range of products that we have available. we do have the lowest price and huge stocks ready for same - day dispatch.

Two million customers can't be wrong !

See our full range at ***malicious url***

B.2.2 Instance 2 (Phishing Email)

Dear Sir / Madam:

Always looks forward for the high security of our clients. Some customers have been receiving phishing emails claiming to be from barclays and advising them to follow a link to what appear to be a barclays web site, where they are prompted to enter their personal online banking details. Barclays is in no way involved with this email and the web site does not belong to us.

For your security, we updated our new ssl servers. Barclays is proud to annouce about the new secure system, which will give our customers a better, fast and secure online banking service.

Due to the recent update of the servers, you are requested to update your account info at the following link.

J.S. Smith. Security Advisor Barclays Bank PLC. Please do not reply to this e - mail . mail sent to this address cannot be answered. For assistance , log in to your barclays online bank account and choose the " help " link on any page.

B.2.3 Instance 3 (Phishing Email)

Because someone has reported your actions, your account will be deactivated. Maybe you have written content that is abusive or uploaded a picture that can be insulting or harmful to other users. You must confirm your account to stop the warning on your account.

To stop the suspension of your account, please click the link below: ***malicious url***

Facebook Game Network Inc.

Phone: 650.543.4800 fax: 650.543.4801

B.2.4 Instance 4 (Spam)

Search Engine Position!

Be the very first listing in the top search engines immediately . Our company will now place any business with a qualified website permanently at the top of the major search engines guaranteed never to move: (E. google, yahoo!, msn, alta vista, etc.).

This promotion includes unlimited traffic and is not going to last long. if you are interested in being guaranteed first position in the top search engines at a promotional fee, please contact us promptly to find out if you qualify via email at searchl1@telefonica.net.pe. It's very important to include the url (s) if you are interested in promoting ! ! ! this is not pay per click. Examples will be provided.

P.S. This promotion is only valid in the usa and canada.

Sincerely ,

The Search Engine Placement Specialists.

If you wish to be removed from this list, please respond to the following email address and type the word "remove" in your subject line: search6@speedy.com.pe

B.2.5 Instance 5 (Phishing Email)

Greetings,

After reviewing your LinkedIn profile, our company would like to present you a part-time job offer as a finance officer in your region. This job does not require any previous experience. Here is a list of tasks that our employee should accomplish:

1. Receive payment from our customers into your bank account

2. Keep your commission fee of 10% from the payment amount

3. Send the rest of the payment to one of our payment receivers in Europe via Moneygram or Western Union.

For more details about this job offer, click ***here***

After enrolling to our part-time job you will be contacted by one of our human resource staff.

Thanks.

Karen Hoffman,

Human Resource Manager.

B.2.6 Instance 6 (Phishing Email)

Bank of America Online Banking

Message from Customer Service

To stop the suspension of your account, please click the link below: ***http://www.facebook.com/account-suspend/*** *A message from Customer Service is waiting in your Online Banking mailbox. If you havent already read it:*

Sign in to Online Banking at ***malicious url***

Select Mail at the top of the page.

This alert relates to your Online Banking profile, rather than a particular account.

Want to confirm this email is from Bank of America? Sign in to Online Banking and select Alerts History to verify this alert.

Want to get more alerts? Sign in to your online banking account at Bank of America and within the Accounts Overview page select the "Alerts" tab.

Because email is not a secure form of communication, this email box is not equipped to handle replies.

If you have any questions about your account or need assistance, please call the phone number on your statement or go to Contact Us at ***www.bankofamerica.com****.*

B.2.7 Instance 7 (Spam)

Discover you made money while you were sleeping!

You must read this word for word ! Information that you may not receive again so please take it seriously ! ! Would you like to . . . . receive thousands in cash daily?

If yes, go here now !

People are making real fortunes, no hype - no false predictions ! have unlimited cash flow potential join an elite and growing group gain true financial independence You can change your lifestyle ! and we can prove it ! !

Our generating leveraging system has been proven 100% effective. Totally duplicable for anyone! The serious money is right here ! ! Do yourself a favor and take a close look at this, you' ll be thankful you did !

This is not sales, our private website will give you all the details. Go here to get them

now !

If you received this by error or wish to be excused from our list, simply click here.

B.2.8 Instance 8 (Phishing Email)

Dear Customer,

You have received this email because we have strong reason to believe that your Amazon account had been recently compromised. In order to prevent any fraudulent activity from occurring we are required to open an investigation into this matter.

If your account is not confirmed, we reserve the right to terminate your Amazon subscription. If you received this notice and you are not an authorized Amazon account holder, please be aware that it is in violation of Amazon policy to present oneself as an Amazon user. Such action may also be in violation of local, national, and/or international law. Amazon is committed to assist law enforcement with any inquires related to attempts to misappropriate personal information with the intent to commit fraud or theft. Information will be provided at the request of law enforcement agencies to ensure that perpetrators are prosecuted to the full extent of the law.

To confirm your identity with us click the link bellow: ***malicious url***

We apologize in advance for any inconvenience this may cause you and we would like to thank you for your cooperation as we review this matter.

B.2.9 Instance 9 (Phishing Email)

Dear Taxpayer,

I am sending this email to announce: After the last annual calculation of your fiscal activity, we have determined that you are eligible to receive a tax return of: $273.48

In order for us to return the excess payment, you need to create a e-Refund account after which the funds will be credited to your specified bank account.

Please click ***Get Started*** *to claim your refund:*

B.2.10 Instance 10 (Spam)

Ink prices got you down?

*Would you like to save up to 80 % on printer ,fax and copier supplies? On brands like epson, canon, hewlett, packard, lexmark and more ! 100 % quality satisfaction guarantee or your money back ! Free same day shipping on all us orders ! We'll beat any price on the internet - guaranteed ! * * Click here to order now ! or Call us toll - free at 1 - 800 - 758 - 8084 ! * Free shipping only on orders of $ 40 or more . * * We beat any online retailer's price by 5 % . * Call us with any other source advertising a lower price and once we verify the price, we will beat it by 5 %! (must be same manufacturer) You are receiving this special offer because you have provided permission to receive email communications regarding special online promotions or offers . If you feel you have received this message in error , or wish to be removed from our subscriber list , Click Here.*

Thank you and we apologize for any inconvenience.

B.3 Additional Material about the Impact of Randomization

While randomization has a significant effect on the ability of subjects to evade email as described in the main document, we found little evidence of impact on either score (of non-filtered submissions) or time taken to submit. The score for tasks including randomization and those which did not is shown as a function of submission sequence in Figure B.3. The differences in the scores are not statistically significant, and there does not appear to be any systematic difference or trend with experience (unlike the ability to evade the filter, which demonstrates clear improvement over time).

Similarly, the differences in time spent on a submission are not statistically different between randomized and noise-free settings (Figure B.4). Here, the initial few submissions clearly took the longest, but thereafter the trend is quite weak (although submission time does appear to decrease slightly with experience).

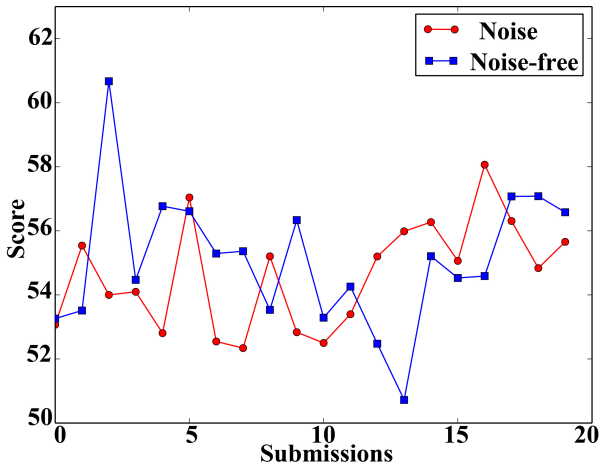


Figure B.3: The scores of non-flipped submissions in the "noise" treatments compared to the scores in the "noise-free" treatment, as a function of the submission sequence. There is no clear difference between the two sets of treatments, or a clear trend.

B.4 Additional Details for the Synthetic Model of Human Evasion Behavior

As described in the main text, our synthetic model has two pieces: (1) the model to predict, for each feature, whether or not it will be changed in the next submission, and (2) if a feature is deleted, whether or not it is substituted for by another.

For model (1), a submission is modeled entirely as a feature vector x corresponding to the features in our classifier (filter). We treat each feature in the submission vector x as independent, and train n independent Support Vector Machine models with a radial basis function kernel [139], one for each submission feature (for features of the subsequent submission we are trying to predict). For each of these, the predicted variable is a binary indicator whether or not the corresponding feature is changed. Features of each of these n models (as opposed to the predicted outputs themselves) include: a) feature vectors corresponding to the two prior submissions ("ideal" email is used for this purpose for the first two submissions), b) gender of the participant, c) participant education level, d) age of the participant, e) English test score of the participant, and f) scores earned by the two prior submissions (the submission is indicated to be filtered by the classifier when the score is 0).

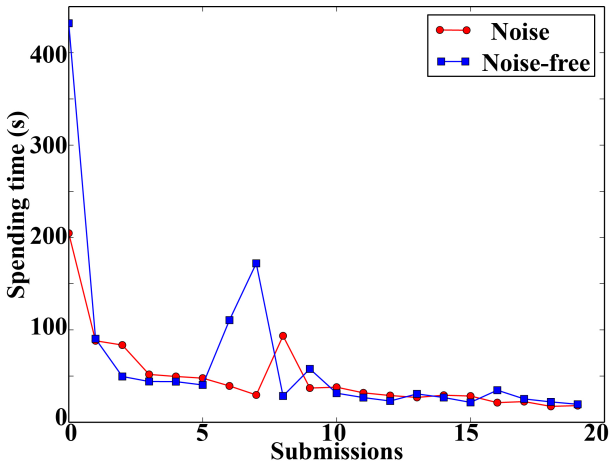


Figure B.4: Time spent on submissions in the "noise" and "noise-free" treatments as a function of the submission sequence. There is little difference between the two treatments (the difference is not significant), and only a weak trend (submission time is decreasing with experience).

For model (2), we train a Support Vector Machine model for each deleted word to determine whether or not it is substituted (again, a binary classification problem). We use the same feature vector as for model (1), as well as two additional features, each corresponding to a binary indicator whether or not a specific feature word was substituted in the two prior submissions. For the first two submissions, these two features represent whether or not the word is a "substituted word" based on the training data information. We define the Substituted Ratio of a word as $S_R = \frac{N_{sub}}{N_{del}}$, where N_{sub} and N_{del} correspond to the number of substitution and deletion times of the word within the training submissions, respectively. If $S_R > 0.5$, the feature word is considered as a "substituted word", and the corresponding feature value is 1; otherwise 0.

Figure B.5 shows accuracy as a function of the submission sequence (the overall accuracy was 97%). Figures B.6 and B.7 show average score as a function of the submission sequence in the noise-free and noisy treatments, respectively, comparing the performance for the two scoring functions, S_1 and S_2. In all, the evidence clearly indicates that our synthetic model performs extremely well both in predicting individual behavior as well as in synthetically replicating experimental observables.

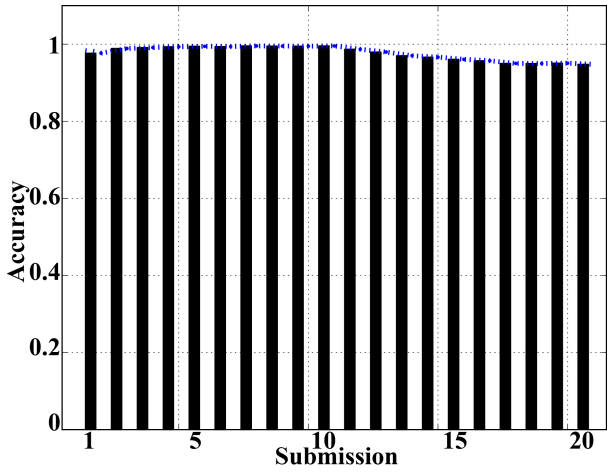


Figure B.5: The average cross-validation accuracy shows that the synthetic model is able to predict the next submission vector based on the previous two submissions accurately.

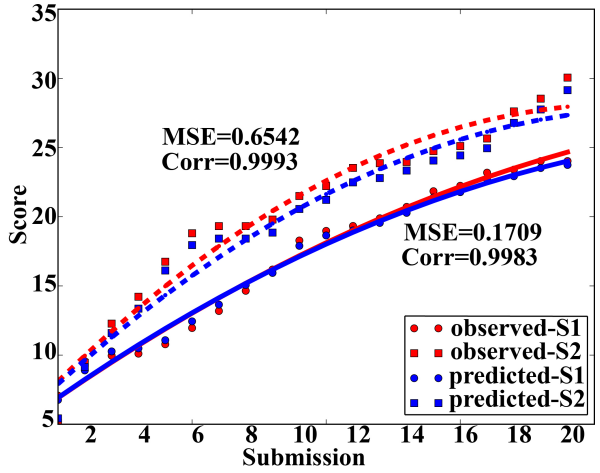


Figure B.6: Comparison between experimentally observed scores and those based on the synthetic model of behavior as a function of the submission sequence for S_1 and S_2 scoring function treatments in the "noise free" setting.

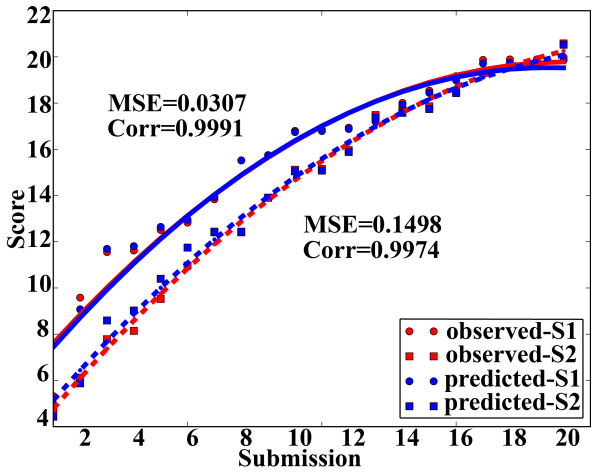
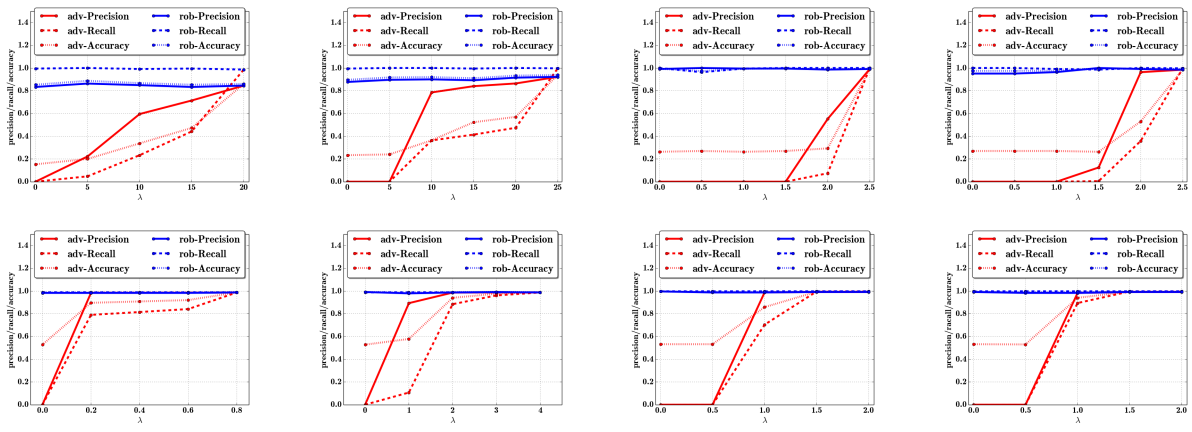


Figure B.7: Comparison between experimentally observed scores and those based on the synthetic model of behavior as a function of the submission sequence for S_1 and S_2 scoring function treatments when filter randomization was used.

Appendix C

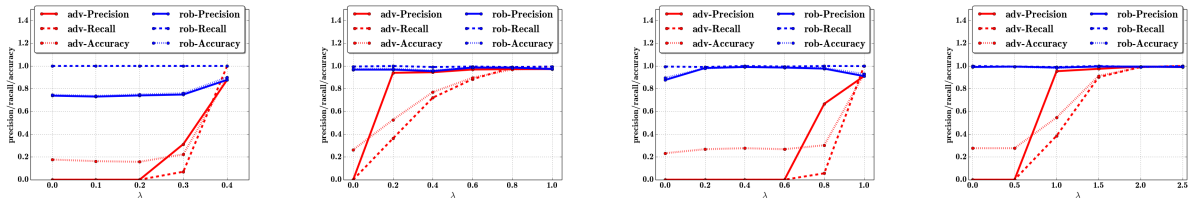
Appendix for chapter 7

C.1 Supplemental comparison results based on trimmed size of features



(a) (b) (c) (d)

Figure C.1: Performance of baseline (*adv-*) and *RAD* (*rob-*) as a function of cost sensitivity λ for Enron (top) and MNIST (bottom) datasets with continuous features testing on adversarial instances based on 1000 (Enron) and 627 (MNIST) features. (a) logistic regression, (b) SVM, (c) 1-layer NN, (d) 3-layer NN.



(a) (b) (c) (d)

Figure C.2: Performance of baseline (*adv-*) and *RAD* (*rob-*) implementations of (a) Naive Bayes, (b) logistic regression, (c) SVM, and (d) 3-layer NN, using binary features testing on adversarial instances based on 1000 features.

Appendix D

Appendix for chapter 8

D.1 Gradient optimization for attack strategies

We provide details on how to compute the "easy" gradient $\nabla_{\widetilde{\Theta}} R(\widehat{M}, M)$, $\widehat{M} = \widehat{M}(\Theta)$ is the prediction based on the learnt model Θ. Applying the chain rule of differentiation we get

$$\nabla_{\Theta} R(\widehat{M}, M) = \left(\nabla_{\Theta} \widehat{M}\right) \left(\nabla_{\widehat{M}} R(\widehat{M}, M)\right). \tag{D.1}$$

We first focus on the second term $\nabla_{\widehat{M}} R(\widehat{M}, M))$. This is easy to compute because all malicious utility functions R considered here are smooth and differentiable. More specifically, the availability attack utility R^{av} and the integrity attack utility R^{in} admit the following gradient computations:

$$\begin{aligned} \frac{\partial R^{\text{av}}}{\partial \widehat{M}_{ij}} &= 2(\widehat{M}_{ij} - \overline{M}_{ij}) \cdot I[(i,j) \notin \Omega]; \\ \frac{\partial R^{\text{in}}_{J_0,w}}{\partial \widehat{M}_{ij}} &= w(j) \cdot I[j \in J_0]. \end{aligned}$$

Here $I[\cdot]$ is the indicator function that equals one if the corresponding condition holds true and zero otherwise. The gradient for the hybrid utility R^{hybrid} can then be expressed as a linear combination of the gradients of R^{av} and R^{in}:

$$\nabla R^{\text{hybrid}}_{\mu,J_0,w} = \mu_1 \nabla R^{\text{av}} + \mu_2 \nabla R^{\text{in}}_{J_0,w}.$$

We next turn to the computation of $\nabla_{\Theta} \widehat{M}$, which is model specific. Alternating minimization and nuclear norm minimization are considered separately for this gradient:

Alternating minimization In alternating minimization the learnt model Θ is parame-

terized by $\Theta = (\boldsymbol{U}, \widetilde{\boldsymbol{U}}, \boldsymbol{V})$, where $\boldsymbol{U} \in \mathbb{R}^{m\times k}$, $\widetilde{\boldsymbol{U}} \in \mathbb{R}^{m'\times k}$ and $\boldsymbol{V} \in \mathbb{R}^{n\times k}$. Since $\widehat{\boldsymbol{M}} = \boldsymbol{U}\boldsymbol{V}^\top$ for normal users, we have

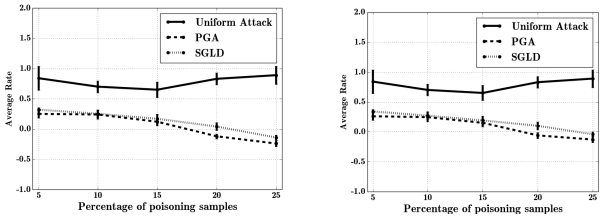
$$\frac{\partial \widehat{\boldsymbol{M}}_{ij}}{\partial \boldsymbol{U}_{\ell t}} = \boldsymbol{V}_{jt} \cdot I[i=\ell], \quad \frac{\partial \widehat{\boldsymbol{M}}_{ij}}{\partial \boldsymbol{V}_{\ell t}} = \boldsymbol{U}_{it} \cdot I[j=\ell].$$

Nuclear norm minimization In nuclear norm minimization the learnt model Θ is parameterized by $\Theta = (\boldsymbol{U}, \widetilde{\boldsymbol{U}}, \boldsymbol{V}, \boldsymbol{\Sigma})$ where $\boldsymbol{U} \in \mathbb{R}^{m\times k}$, $\widetilde{\boldsymbol{U}} \in \mathbb{R}^{m'\times k}$, $\boldsymbol{V} \in \mathbb{R}^{n\times k}$ and $\boldsymbol{\Sigma} = \boldsymbol{diag}(\sigma_1, \cdots, \sigma_k)$. The estimation $\widehat{\boldsymbol{M}}$ for normal users is then expressed as $\widehat{\boldsymbol{M}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$. As a result, we have

$$\begin{aligned}
\frac{\partial \widehat{\boldsymbol{M}}_{ij}}{\partial \boldsymbol{U}_{\ell t}} &= \sigma_t \boldsymbol{V}_{jt} \cdot I[i=\ell]; \\
\frac{\partial \widehat{\boldsymbol{M}}_{ij}}{\partial \boldsymbol{V}_{\ell t}} &= \sigma_t \boldsymbol{U}_{it} \cdot I[j=\ell]; \\
\frac{\partial \widehat{\boldsymbol{M}}_{ij}}{\partial \sigma_t} &= \boldsymbol{U}_{it}\boldsymbol{V}_{jt}.
\end{aligned}$$

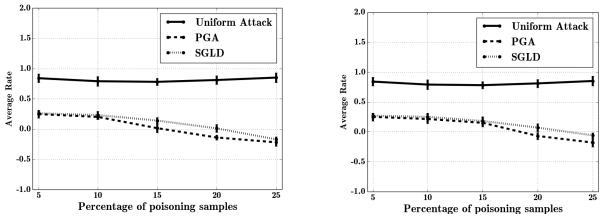
D.2 Supplement results for adversarial decreasing rates

Here we plot ratings of specific items against percentage of malicious profiles by setting $\mu_2 = -1$ to evaluate the performance of attacker reducing the popularity of the item, whose original predicted average rating is 0.8. Figure D.1 and D.2 both show two settings of $\mu_1 = 0, \mu_2 = -1$ and $\mu_1 = -1, \mu_2 = -1$ for alternating minimization and nuclear norm minimization, respectively. For alternating minimization algorithm, when $\mu_1 = 0, \mu_2 = -1$, the attacker tries to reduce the average rating for certain item without caring about the availability error of the whole recommendation system. This way, the attacker has better control of the item and can decrease the average rating of the item from 0.8 to around -0.3. While, if $\mu_1 = -1, \mu_2 = -1$, the attacker want to reduce the popularity of the item and at the same time reduce the availability error for the whole system to avoid detection; therefore the attacker can only decrease the average rating of the item to about -0.1 under



(a) (b)

Figure D.1: Average ratings of certain items using alternating minimization; (a) $\mu_1 = 0, \mu_2 = -1$, (b) $\mu_1 = -1, \mu_2 = -1$.



(a) (b)

Figure D.2: Average ratings of certain items using nuclear norm minimization; (a) $\mu_1 = 0, \mu_2 = -1$, (b) $\mu_1 = -1, \mu_2 = -1$.

this setting.We obtain the similar observations for the nuclear norm minimization.

Appendix E

Appendix for chapter 9

Proof of Lemma 10.5.4. By Theorem 10.5.1, Algorithm 16 will terminate when $\frac{\widehat{FP_A}}{\widehat{TP_A}} \geq \frac{L}{C}$. Using, Equation 10.24 and our assumptions, we have

$$\frac{(1+\frac{1}{p})\widehat{FP_A}}{m} \leq \frac{(1+\frac{1}{s})FP_A}{m} + \lambda(m,\delta)$$

with probability at least $1-\delta$. Consequently, with probability at least $1-\delta$,

$$\frac{(1+\frac{1}{p})\widehat{FP_A}}{TP_A \cdot \widehat{TP_A}} \leq \frac{(1+\frac{1}{s})FP_A}{TP_A\widehat{TP_A}} + \lambda(m,\delta)\frac{m}{\widehat{TP_A}TP_A},$$

and, consequently,

$$\frac{1+\frac{1}{p}}{TP_A} \cdot \frac{L}{C} \leq \frac{(1+\frac{1}{s})FP_A}{TP_A\widehat{TP_A}} + \lambda(m,\delta)\frac{m}{\widehat{TP_A}TP_A}.$$

Rearranging, we get

$$\begin{aligned}
\frac{FP_A}{TP_A} &\geq \frac{(1+\frac{1}{p})\frac{L}{C}\cdot\frac{1}{TP_A} - \lambda(m,\delta)\cdot\frac{m}{\widehat{TP_A}}\cdot\frac{1}{TP_A}}{(1+\frac{1}{s})\cdot\cdot\frac{1}{\widehat{TP_A}}} \\
&= \frac{1}{1+\frac{1}{s}}\left(\frac{L}{C}(1+\frac{1}{p})\cdot\frac{\widehat{TP_A}}{m}\cdot\frac{m}{TP_A} - \lambda(m,\delta)\cdot\frac{N_A}{TP_A}\right) \\
&\geq \left(\frac{1}{1+\frac{1}{s}}\right)\left((1+\frac{1}{p})\frac{\widehat{TP_A}}{m}\frac{L}{C} - \lambda(m,\delta)\right)\frac{m}{TP_A} \\
&\geq \left(\frac{1}{1+\frac{1}{s}}\right)\left((1+\frac{1}{p})\cdot q\cdot\frac{L}{C} - \lambda(m,\delta)\right)\frac{1}{r}
\end{aligned}$$

□

BIBLIOGRAPHY

[1] Daniel Jurafsky and H James. Speech and language processing an introduction to natural language processing, computational linguistics, and speech. 2000.

[2] Wenyi Zhao, Rama Chellappa, P Jonathon Phillips, and Azriel Rosenfeld. Face recognition: A literature survey. *Acm Computing Surveys (CSUR)*, 35(4):399–458, 2003.

[3] Réjean Plamondon and Sargur N Srihari. Online and off-line handwriting recognition: a comprehensive survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1):63–84, 2000.

[4] Peter Bodik, 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*, pages 241–252. ACM, 2010.

[5] Wei Xu, Peter Bodik, and David Patterson. A flexible architecture for statistical learning and data mining from system log streams. *Temporal Data Mining: Algorithms, Theory and Applications, Brighton, UK*, 2004.

[6] 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*, volume 37. ACM, 2007.

[7] Srikanth Kandula, Ranveer Chandra, and Dina Katabi. What's going on?: learning communication rules in edge networks. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 87–98. ACM, 2008.

[8] Tony A Meyer and Brendon Whateley. Spambayes: Effective open-source, bayesian based, email classification system. In *CEAS*. Citeseer, 2004.

[9] Tom Fawcett and Foster Provost. Adaptive fraud detection. *Data mining and knowledge discovery*, 1(3):291–316, 1997.

[10] Ted E Senator. Ongoing management and application of discovered knowledge in a large regulatory organization: a case study of the use and impact of nasd regulation's advanced detection system(rads). In *Conference on Knowledge Discovery in Data: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, volume 20, pages 44–53, 2000.

[11] Matthew V Mahoney and Philip K Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 376–385. ACM, 2002.

[12] Robert B Doorenbos, Oren Etzioni, and Daniel S Weld. A scalable comparison-shopping agent for the world-wide web. In *Proceedings of the first international conference on Autonomous agents*, pages 39–48. ACM, 1997.

[13] Colin Whittaker, Brian Ryner, and Marria Nazif. Large-scale automatic classification of phishing pages. In *NDSS*, volume 10, 2010.

[14] Charlie Curtsinger, Benjamin Livshits, Benjamin G Zorn, and Christian Seifert. Zozzle: Fast and precise in-browser javascript malware detection. In *USENIX Security Symposium*, pages 33–48, 2011.

[15] Paul Graham. A plan for spam, 2002.

[16] Michael Wooldridge. Does game theory work? *Intelligent Systems, IEEE*, 27(6):76–80, 2012.

[17] George Cybenko and Carl E Landwehr. Security analytics and measurements. *IEEE Security & Privacy*, 10(3):0005–8, 2012.

[18] Christoph Sawade, Tobias Scheffer, et al. Bayesian games for adversarial regression problems. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 55–63, 2013.

[19] Huan Xu, Constantine Caramanis, and Shie Mannor. Robust regression and lasso. In *Advances in Neural Information Processing Systems*, pages 1801–1808, 2009.

[20] Adam Barth, Benjamin IP Rubinstein, Mukund Sundararajan, John C Mitchell, Dawn Song, and Peter L Bartlett. A learning-based approach to reactive security. In *Financial Cryptography and Data Security*, pages 192–206. Springer, 2010.

[21] Battista Biggio, Igino Corona, Blaine Nelson, Benjamin IP Rubinstein, Davide Maiorca, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli. Security evaluation of support vector machines in adversarial environments. In *Support Vector Machines Applications*, pages 105–153. Springer, 2014.

[22] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[23] Vladimir Vapnik. *The nature of statistical learning theory*. springer, 2000.

[24] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45, 1997.

[25] Marco Barreno, Blaine Nelson, Anthony D Joseph, and JD Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.

[26] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25. ACM, 2006.

[27] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, 4:169–191, 1883.

[28] Chad Perrin. The cia triad. *Dostopno na: http://www. techrepublic. com/blog/security/the-cia-triad/488*, 2008.

[29] Daniel C Barth-Jones. The "re-identification" of governor william weld's medical.

[30] Fida K Dankar, Khaled El Emam, Angelica Neisa, and Tyson Roffey. Estimating the re-identification risk of clinical data sets. *BMC medical informatics and decision making*, 12(1):66, 2012.

[31] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[32] 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*, pages 59–68. ACM, 2006.

[33] Amir Globerson and Sam Roweis. Nightmare at test time: robust learning by feature deletion. In *Proceedings of the 23rd international conference on Machine learning*, pages 353–360. ACM, 2006.

[34] Matthew V Mahoney and Philip K Chan. An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In *Recent Advances in Intrusion Detection*, pages 220–237. Springer, 2003.

[35] Kymie MC Tan, Kevin S Killourhy, and Roy A Maxion. Undermining an anomaly-based intrusion detection system using common exploits. In *Recent Advances in Intrusion Detection*, pages 54–73. Springer, 2002.

[36] Anil Somayaji and Stephanie Forrest. Automated response using system-call delay. In *Usenix Security Symposium*, pages 185–197, 2000.

[37] Kymie Tan, John McHugh, and Kevin Killourhy. Hiding intrusions: From the abnormal to the normal and beyond. In *Information Hiding*, pages 1–17. Springer, 2003.

[38] 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*, pages 255–264. ACM, 2002.

[39] Daniel Lowd and Christopher Meek. Good word attacks on statistical spam filters. In *CEAS*, 2005.

[40] Gregory L Wittel and Shyhtsun Felix Wu. On attacking statistical spam filters. In *CEAS*, 2004.

[41] 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. ACM, 2007.

[42] David Sculley, Gabriel Wachman, and Carla E Brodley. Spam filtering using inexact string matching in explicit feature space with on-line linear classifiers. In *TREC*, 2006.

[43] Bo Li and Yevgeniy Vorobeychik. Feature cross-substitution in adversarial classification. In *Advances in Neural Information Processing Systems*, pages 2087–2095, 2014.

[44] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108. ACM, 2004.

[45] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647. ACM, 2005.

[46] 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.

[47] Mark Dredze, Reuven Gevaryahu, and Ari Elias-Bachrach. Learning fast classifiers for image spam. In *CEAS*, 2007.

[48] WANG Zhe, W Josephson, LV Qin, et al. Filtering image spam with near-duplicate detection. In *Proc of the 4th Conference on E-mail and AntiSpam*, 2007.

[49] James Newsome, Brad Karp, and Dawn Song. Paragraph: Thwarting signature learning by training maliciously. In *Recent advances in intrusion detection*, pages 81–105. Springer, 2006.

[50] Shobha Venkataraman, Avrim Blum, and Dawn Song. Limits of learning-based signature generation with adversaries. 2008.

[51] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 219–230. ACM, 2004.

[52] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and JD Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 1–14. ACM, 2009.

[53] Simon P Chung and Aloysius K Mok. Allergy attack against automatic signature generation. In *Recent Advances in Intrusion Detection*, pages 61–80. Springer, 2006.

[54] Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *USENIX security symposium*, volume 286. San Diego, CA, 2004.

[55] Charu C Aggarwal, Jian Pei, and Bo Zhang. On privacy preservation against adversarial data mining. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 510–516. ACM, 2006.

[56] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3, 2007.

[57] Traian Marius Truta and Bindu Vinay. Privacy protection: p-sensitive k-anonymity property. In *ICDE Workshops*, page 94, 2006.

[58] Gagan Aggarwal, Tomas Feder, Krishnaram Kenthapadi, Samir Khuller, Rina Panigrahy, Dilys Thomas, and An Zhu. Achieving anonymity via clustering. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 153–162. ACM, 2006.

[59] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, volume 7, pages 106–115, 2007.

[60] Xiaokui Xiao and Yufei Tao. M-invariance: towards privacy preserving republication of dynamic datasets. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 689–700. ACM, 2007.

[61] Mehmet Ercan Nergiz, Maurizio Atzori, and Chris Clifton. Hiding the presence of individuals from shared databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 665–676. ACM, 2007.

[62] Oscar Ferrández, Brett R South, Shuying Shen, F Jeffrey Friedlin, Matthew H Samore, and Stéphane M Meystre. Bob, a best-of-breed automated text de-identification system for vha clinical documents. *Journal of the American Medical Informatics Association*, 20(1):77–83, 2013.

[63] Stéphane Meystre, Shuying Shen, Deborah Hofmann, and Adi Gundlapalli. Can physicians recognize their own patients in de-identified notes? *Studies in health technology and informatics*, 205:778, 2014.

[64] Vijay S Iyengar. Transforming data to satisfy privacy constraints. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 279–288. ACM, 2002.

[65] Adam Meyerson and Ryan Williams. On the complexity of optimal k-anonymity. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 223–228. ACM, 2004.

[66] Roberto J Bayardo and Rakesh Agrawal. Data privacy through optimal k-anonymization. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 217–228. IEEE, 2005.

[67] Charu C Aggarwal. On k-anonymity and the curse of dimensionality. In *Proceedings of the 31st international conference on Very large data bases*, pages 901–909. VLDB Endowment, 2005.

[68] Elisa Bertino, Beng Chin Ooi, Yanjiang Yang, and Robert H Deng. Privacy and ownership preserving of outsourced medical data. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 521–532. IEEE, 2005.

[69] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 49–60. ACM, 2005.

[70] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. Workload-aware anonymization. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 277–286. ACM, 2006.

[71] Ke Wang and Benjamin Fung. Anonymizing sequential releases. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 414–423. ACM, 2006.

[72] Daniel Kifer and Johannes Gehrke. Injecting utility into anonymized datasets. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 217–228. ACM, 2006.

[73] Xiaokui Xiao and Yufei Tao. Anatomy: Simple and effective privacy preservation. In *Proceedings of the 32nd international conference on Very large data bases*, pages 139–150. VLDB Endowment, 2006.

[74] Qing Zhang, Nick Koudas, Divesh Srivastava, and Ting Yu. Aggregate query answering on anonymized tables. In *ICDE*, volume 7, pages 116–125. Citeseer, 2007.

[75] Ke Wang, Janak J Parekh, and Salvatore J Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Advances in Intrusion Detection*, pages 226–248. Springer, 2006.

[76] Yevgeniy Vorobeychik and Bo Li. Optimal randomized classification in adversarial settings. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 485–492. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[77] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D Joseph, Benjamin IP Rubinstein, Udam Saini, Charles Sutton, JD Tygar, and Kai Xia. Misleading learners: Co-opting your spam filter. In *Machine learning in cyber trust*, pages 17–51. Springer, 2009.

[78] Michael Kearns and Ming Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837, 1993.

[79] Peter J Huber. *Robust statistics*. Springer, 2011.

[80] David E Tyler. Robust statistics: Theory and methods. *Journal of the American Statistical Association*, 103(482):888–889, 2008.

[81] David Wagner. Resilient aggregation in sensor networks. In *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 78–87. ACM, 2004.

[82] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[83] Wei Liu and Sanjay Chawla. Mining adversarial patterns via regularized loss minimization. *Machine learning*, 81(1):69–83, 2010.

[84] Michael Brückner and Tobias Scheffer. Stackelberg games for adversarial prediction problems. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 547–555. ACM, 2011.

[85] MohamadAli Torkamani and Daniel Lowd. Convex adversarial collective classification. In *Proceedings of The 30th International Conference on Machine Learning*, pages 642–650, 2013.

[86] Yan Zhou, Murat Kantarcioglu, Bhavani Thuraisingham, and Bowei Xi. Adversarial support vector machine learning. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1059–1067. ACM, 2012.

[87] Yan Zhou and Murat Kantarcioglu. Adversarial learning with bayesian hierarchical mixtures of experts.

[88] Security and Response Group. Internet security threat report, 2015.

[89] Duen Horng Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. Polonium: Tera-scale graph mining and inference for malware detection. In *SIAM International Conference on Data Mining*, volume 2, 2011.

[90] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. Guilt by association: large scale malware detection by mining file-relation graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1524–1533. ACM, 2014.

[91] Jin Zhang and Qian Zhang. Stackelberg game for utility-based cooperative cognitiveradio networks. In *Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing*, pages 23–32. ACM, 2009.

[92] Frederic Perriot and Peter Ferrie. Principles and practise of x-raying. In *Virus Bulletin Conference*, Chicago, IL, 2004.

[93] Fanglu Guo, Peter Ferrie, and Tzicker Chiueh. A study of the packer problem and its solutions. In *Symposium on Recent Advances in Intrusion Detection (RAID)*, Cambridge, MA, 2008. Springer Berlin / Heidelberg.

[94] Lorenzo Martignoni, Mihai Christodorescu, and Somesh Jha. Omniunpack: Fast, generic, and safe unpacking of malware. In *Annual Computer Security Applications Conference (ACSAC)*, Miami Beach, FL, 2007.

[95] Xin Hu, Kang G. Shin, Sandeep Bhatkar, and Kent Griffin. Mutantx-s: Scalable malware clustering based on static features. In *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 187–198, San Jose, CA, 2013. USENIX.

[96] Jiyong Jang, David Brumley, and Shobha Venkataraman. Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 309–320. ACM, 2011.

[97] Grégoire Jacob, Hervé Debar, and Eric Filiol. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in computer Virology*, 4(3):251–266, 2008.

[98] Roberto Perdisci, Wenke Lee, and Nick Feamster. Building a dynamic reputation system for dns. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, 2010.

[99] Manoj Parameswaran, Huaxia Rui, and S Sayin. A game theoretic model and empirical analysis of spammer strategies. In *Collaboration, Electronic Messaging, AntiAbuse and Spam Conf*, volume 7, 2010.

[100] Michael Brückner and Tobias Scheffer. Nash equilibria of static prediction games. In *Advances in neural information processing systems*, pages 171–179, 2009.

[101] Tudor Dumitras and Darren Shou. Toward a standard benchmark for computer security research: the worldwide intelligence network environment (wine). In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, Salzburg, Austria, 2011.

[102] Kevin A. Roundy and Barton P. Miller. Binary-code obfuscations in prevalent packer tools. *ACM Computing Surveys (CSUR)*, 46(1), 2013.

[103] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[104] Mantas LukošEvičIus and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[105] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, Edinburgh, Scotland, UK, 1999.

[106] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586. ACM, 2011.

[107] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010.

[108] Huan Liu and Rudy Setiono. Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of 7th Intenational Conference on Tools with Artificial Intelligence (ICTAI)*, pages 388–391.

[109] Microsoft Corporation. Microsoft portable executable and common object file format specification. 1999. Revision 6.0.

[110] Tomas Mikolov, Stefan Kombrink, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (ICASSP)*, Prague, Czech Republic, 2011.

[111] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[112] J. Zico Kolter and Marcus A. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7:2721–2744, 2006.

[113] Bo Li and Yevgeniy Vorobeychik. Scalable optimization of randomized operational decisions in adversarial classification settings. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2015. in press.

[114] Yevgeniy Vorobeychik and Bo Li. Optimal randomized classification in adversarial settings. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 485–492, 2014.

[115] Bo Li and Yevgeniy Vorobeychik. Feature cross-substitution in adversarial classification. In *Neural Information Processing Systems*, 2014. to appear.

[116] Marco Barreno, Peter L Bartlett, Fuching Jack Chi, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, Udam Saini, and J Doug Tygar. Open problems in the security of learning. In *Proceedings of the 1st ACM workshop on Workshop on AISec*, pages 19–26. ACM, 2008.

[117] Battista Biggio, Giorgio Fumera, and Fabio Roli. Security evaluation of pattern classifiers under attack. 2013.

[118] Pavel Laskov and Richard Lippmann. Machine learning in adversarial environments. *Machine learning*, 81(2):115–119, 2010.

[119] Blaine Nelson, Benjamin IP Rubinstein, Ling Huang, Anthony D Joseph, and JD Tygar. Classifier evasion: Models and open problems. In *Privacy and Security Issues in Data Mining and Machine Learning*, pages 92–98. Springer, 2011.

[120] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop*, volume 62, pages 98–105, 1998.

[121] KONG Ying and ZHAO Jie. Learning to filter unsolicited commercial e-mail. *International Proceedings of Computer Science & Information Technology*, 49, 2012.

[122] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. Spam filtering with naive bayes-which naive bayes? In *CEAS*, pages 27–28, 2006.

[123] Laurent El Ghaoui, Gert René Georges Lanckriet, Georges Natsoulis, et al. *Robust classification with interval data*. Computer Science Division, University of California, 2003.

[124] Wei Liu and Sanjay Chawla. A game theoretical model for adversarial learning. In *Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on*, pages 25–30. IEEE, 2009.

[125] Tom Fawcett. In vivo spam filtering: a challenge problem for kdd. *ACM SIGKDD Explorations Newsletter*, 5(2):140–148, 2003.

[126] Ion Androutsopoulos, Evangelos F Magirou, and Dimitrios K Vassilakis. A game theoretic model of spam e-mailing. In *CEAS*, 2005.

[127] Tiago A Almeida, Akebo Yamakami, and Jurandy Almeida. Evaluation of approaches for dimensionality reduction applied with naive bayes anti-spam filters. In *Machine Learning and Applications, 2009. ICMLA'09. International Conference on*, pages 517–522. IEEE, 2009.

[128] B. Nelson, B. Rubinstein, L. Huang, A. Joseph, S. Lee, S. Rao, and J. D. Tygar. Query strategies for evading convex-inducing classifiers. *Journal of Machine Learning Research*, 13:1293–1332, 2012.

[129] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *Machine learning: ECML 2004*, pages 217–226. Springer, 2004.

[130] Ion Androutsopoulos, John Koutsias, Konstantinos V Chandrinos, George Paliouras,

and Constantine D Spyropoulos. An evaluation of naive bayesian anti-spam filtering. *arXiv preprint cs/0006013*, 2000.

[131] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[132] Garth P McCormick. Computability of global solutions to factorable nonconvex programs: Part iconvex underestimating problems. *Mathematical programming*, 10(1):147–175, 1976.

[133] Norman H Nie, Alberto Simpser, Irene Stepanikova, and Lu Zheng. Ten years after the birth of the internet, how do americans use the internet in their daily lives. *Stanford Institute for the Quantitative Study of Society*, 2005.

[134] Tom Zeller Jr. Law barring junk e-mail allows a flood instead. *The New York Times*, 1, 2005.

[135] Justin Mason. The spamassassin homepage. *http. V/Spamassassin. org/index. html*, 2004.

[136] Georgios Sakkis, Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Constantine D Spyropoulos, and Panagiotis Stamatopoulos. A memory-based approach to anti-spam filtering for mailing lists. *Information Retrieval*, 6(1):49–73, 2003.

[137] Xavier Carreras and Lluis Marquez. Boosting trees for anti-spam email filtering. *arXiv preprint cs/0109015*, 2001.

[138] Joshua Goodman, Gordon V. Cormack, and David Heckerman. Spam and the ongoing battle for the inbox. *Communications of the ACM*, 50(2):25–33, 2007.

[139] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[140] P. Paruchuri, J. Pearce, Janus Marecki, and Milind Tambe. Playing games for security: An efficient exact algorithm for solving Bayesian Stackelberg games. In

Seventh International Conference on Autonomous Agents and Multiagent Systems, pages 895–902, 2008.

[141] James Pita, Manish Jain, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Using game theory for los angeles airport security. *AI Magazine*, 30(1):43–57, 2009.

[142] James Pita, Manish Jain, Fernando Ordonez, Milind Tambe, and Sarit Kraus. Robust solutions to stackelberg games: Addressing bounded rationality and limited observations in human cognition. *Artificial Intelligence Journal*, 174(15):1142–1171, 2010.

[143] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller. Introduction to wordnet: An on-line lexical database*. *International journal of lexicography*, 3(4):235–244, 1990.

[144] Luis Von Ahn and Laura Dabbish. Designing games with a purpose. *Communications of the ACM*, 51(8):58–67, 2008.

[145] John J Horton, David G Rand, and Richard J Zeckhauser. The online laboratory: Conducting experiments in a real labor market. *Experimental Economics*, 14(3):399–425, 2011.

[146] Siddharth Suri and Duncan J Watts. Cooperation and contagion in web-based, networked public goods experiments. *PLoS One*, 6(3):e16836, 2011.

[147] Winter Mason and Duncan J Watts. Collaborative learning in networks. *Proceedings of the National Academy of Sciences*, 109(3):764–769, 2012.

[148] Richard Colbaugh and Kristin Glass. Predictive defense against evolving adversaries. In *IEEE International Conference on Intelligence and Security Informatics*, pages 18–23, 2012.

[149] Sushil Jajodia, Anup K. Ghosh, V.S. Subrahmanian, Vipin Swarup, Cliff Wang, and X. Sean Wang, editors. *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*. Springer, 2012.

[150] Prahlad Fogla, Monirul I Sharif, Roberto Perdisci, Oleg M Kolesnikov, and Wenke Lee. Polymorphic blending attacks. In *USENIX Security*, 2006.

[151] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58. ACM, 2011.

[152] Aleksander Kołcz and Choon Hui Teo. Feature weighting for improved classifier robustness. In *CEAS09: sixth conference on email and anti-spam*, 2009.

[153] Christoph Karlberger, Günther Bayler, Christopher Kruegel, and Engin Kirda. Exploiting redundancy in natural language to penetrate bayesian spam filters. *WOOT*, 7:1–7, 2007.

[154] Blaine Nelson, Benjamin IP Rubinstein, Ling Huang, Anthony D Joseph, Steven J Lee, Satish Rao, and JD Tygar. Query strategies for evading convex-inducing classifiers. *The Journal of Machine Learning Research*, 13(1):1293–1332, 2012.

[155] Bo Li and Yevgeniy Vorobeychik. Scalable optimization of randomized operational decisions in adversarial classification settings. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 599–607, 2015.

[156] Choon Hui Teo, Amir Globerson, Sam T Roweis, and Alex J Smola. Convex learning with invariances. In *NIPS*, volume 20, pages 1489–1496, 2007.

[157] Michael Brückner, Christian Kanzow, and Tobias Scheffer. Static prediction

games for adversarial learning problems. *Journal of Machine Learning Research*, 13(1):2617–2654, 2012.

[158] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[159] A. Kantchelian, J. D. Tygar, and A. D. Joseph. Evasion and hardening of tree ensemble classifiers. arXiv pre-print, 2015.

[160] Yann LeCun and Corinna Cortes. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2010.

[161] Blind authors. Blind title. In *AAAI Conference on Artificial Intelligence*, 2016.

[162] Zhi-Quan Luo and Paul Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992.

[163] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[164] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, AISec '11, pages 43–58, New York, NY, USA, 2011. ACM.

[165] Daniel Lowd and Christopher Meek. Adversarial learning. In *ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 641–647, 2005.

[166] Michael Brückner and Tobias Scheffer. Nash equilibria of static prediction games. In *Advances in Neural Information Processing Systems*, pages 171–179, 2009.

[167] Amir Globerson and Sam Roweis. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, New York, NY, USA.

[168] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy*, pages 305–316, 2010.

[169] Manish Jain, Jason Tsai, James Pita, Christopher Kiekintveld, S. Rathi, Milind Tambe, and Fernando Ordonez. Software assistants for randomized patrol planning for the lax airport police and the federal air marshals service. *Interfacs*, 40:267–290, 2010.

[170] Ondrej Vanek, Zhengyu Yin, Manish Jain, Branislav Bosansky, Milind Tambe, and Michal Pechoucek. Game-theoretic resource allocation for malicious packet detection in computer networks. In *Eleventh International Conference on Autonomous Agents and Multiagent Systems*, 2012.

[171] Battista Biggio, Giorgio Fumera, and Fabio Roli. Adversarial pattern classification using multiple classifiers and randomisation. In *Lecture Notes in Computer Science*, pages 500–509, 2008.

[172] Jeff Kahn, Gil Kalai, and Nathan Linial. The influence of variables on boolean functions. In *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pages 68–80. IEEE, 1988.

[173] Stas Filshtinskiy. Cybercrime, cyberweapons, cyber wars: Is there too much of it in the air? *Communications of the ACM*, 56(6):28–30, 2013.

[174] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 99–108, New York, NY, USA, 2004. ACM.

[175] Ryan O'Donnell. Some topics in analysis of Boolean functions. In *ACM Symposium on Theory of Computing*, pages 569–578, 2008.

[176] Ronald De Wolf. A brief introduction to fourier analysis on the boolean cube. *Theory of Computing, Graduate Surveys*, 1:1–20, 2008.

[177] Jun Wang, Arjen de Vires, and Marcel Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *SIGIR*, 2006.

[178] Emmanuel Candès and Ben Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2007.

[179] Jian-Feng Cai, Emmanuel Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[180] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. Effective attack models for shilling item-based collaborative filtering systems. In *Proceedings of the 2005 WebKDD Workshop, held in conjuction with ACM SIGKDD2005*, 2005.

[181] Michael P OMahony, Neil J Hurley, and Guenole CM Silvestre. Promoting recommendations: An attack on collaborative filtering. In *Database and Expert Systems Applications*, pages 494–503. Springer, 2002.

[182] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. Low-rank matrix completion using alternating minimization. In *STOC*, 2013.

[183] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. Is feature selection secure against training data poisoning. In *ICML*, 2015.

[184] Shike Mei and Xiaojin Zhu. The security of latent dirichlet allocation. In *AISTATS*, 2015.

[185] Shike Mei and Xiaojin Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *AAAI*, 2015.

[186] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011.

[187] Mehran Mozaffari Kermani, Susmita Sur-Kolay, Anand Raghunathan, and Niraj K Jha. Systematic poisoning attacks on and defenses for machine learning in healthcare. *IEEE Journal of Biomedical and Health Informatics*, 19(6):1893–1905, 2014.

[188] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *ICML*, 2012.

[189] Scott Alfeld, Xiaojin Zhu, and Paul Barford. Data poisoning attacks against autoregressive models. In *AAAI*, 2016.

[190] Olga Klopp, Karim Lounici, and Alexandre Tsybakov. Robust matrix completion. *arXiv:1412.8132*, 2014.

[191] Yudong Chen, Huan Xu, Constantine Caramanis, and Sujay Sanghavi. Robust matrix completion and corrupted columns. In *ICML*, 2011.

[192] Yudong Chen, Ali Jalali, Sujay Sanghavi, and Constantine Caramanis. Low-rank matrix recovery from errors and erasures. *IEEE Transactions on Information Theory*, 59(7):4324–4337, 2013.

[193] Feiping Nie, Hua Wang, Xiao Cai, Heng Huang, and Chris Ding. Robust matrix completion via joint schatten p-norm and lp-norm minimization. In *ICDM*, 2012.

[194] Yu-Xiang Wang and Huan Xu. Stability of matrix factorization for collaborative filtering. In *ICML*, 2012.

[195] Hui Yu and Fei Zhang. Collaborative filtering recommender system in adversarial environment. In *ICMLC*, 2012.

[196] Research GroupLens. *www.grouplens.org*.

[197] U.S. Dept. of Health and Human Services. Standards for privacy and individually identifiable health information; final rule. *Federal Register*, 65(250):82462–82829, 2000.

[198] European Parliament and Council of the European Union. Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal L*, 281:0031–0050, 1995.

[199] Benjamin Fung, Ke Wang, Rui Chen, and Philip S Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (CSUR)*, 42(4):14, 2010.

[200] Cynthia Dwork. Differential privacy: A survey of results. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation*, pages 1–19, 2008.

[201] Latanya Sweeney. Achieving *k*-anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):571–588, 2002.

[202] Yeye He and Jeffrey F. Naughton. Anonymization of set-valued data via top-down, local generalization. *Proceedings of the VLDB Endowment*, 2(1):934–945, 2009.

[203] Giorgos Poulis, Aris Gkoulalas-Divanis, Grigorios Loukides, Spiros Skiadopoulos, and Christos Tryfonopoulos. SECRETA: A system for evaluating and comparing

relational and transaction anonymization algorithms. In *Proceedings of the 17th International Conference on Extending Database Technology*, pages 620–623, 2014.

[204] Giorgos Poulis, Grigorios Loukides, Aris Gkoulalas-Divans, and Spiros Skiadopoulous. Anonymizing data with relational and transaction attributes. In *Proceedings of the European Conference on Machine Learning*, pages 353–369, 2013.

[205] Manolis Terrovitis, Nikos Mamoulis, and Panos Kalnis. Privacy-preserving anonymization of set-valued data. *Proceedings of the VLDB Endowment*, pages 115–125, 2008.

[206] Prakash Nadkarni, Lucila Ohno-Machado, and Wendy Chapman. Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 18(5):544–551, 2011.

[207] John Aberdeen, Samuel Bayer, Reyyan Yeniterzi, Ben Wellner, Cheryl Clark, David Hanauer, Bradley Malin, and Lynette Hirschman. The mitre identification scrubber toolkit: design, training, and assessment. *International journal of medical informatics*, 79(12):849–859, 2010.

[208] A. Benton, S. Hill, L. Ungar, A. Chung, C. Leonard, C. Freeman, and J. H. Holmes. A system for de-identifying medical message board text. *BMC Bioinformatics*, 12 Suppl 3:S2, 2011.

[209] Richard Chow, Philippe Golle, and Jessica Staddon. Detecting privacy leaks using corpus-based association rules. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 893–901. ACM, 2008.

[210] James Gardner, Li Xiong, Kanwei Li, and James J Lu. Hide: heterogeneous information de-identification. In *Proceedings of the 12th International Conference on*

Extending Database Technology: Advances in Database Technology, pages 1116–1119. ACM, 2009.

[211] Özlem Uzuner, Yuan Luo, and Peter Szolovits. Evaluating the state-of-the-art in automatic de-identification. *Journal of the American Medical Informatics Association*, 14(5):550–563, 2007.

[212] Amber Stubbs, Christopher Kotfila, and Ozlem Uzuner. Automated systems for the de-identification of longitudinal clinical narratives: Overview of 2014 i2b2/UTHealth shared task Track 1. *Journal of Biomedical Informatics*, page in press, 2015.

[213] Michael Barbaro, Tom Zeller, and Saul Hansell. A face is exposed for aol searcher no. 4417749. *New York Times*, 9(2008):8For, 2006.

[214] Robert Hackett. Jeb bush exposed 13,000 social security numbers. here's where they were hiding. *Forbes*, page Feb 13, 2015.

[215] Information Commissioner's Office. Anonymisation: managing data protection risk code of practice, 2012.

[216] Roger J Bowden and Ah Boon Sim. The privacy bootstrap. *Journal of Business & Economic Statistics*, 10(3):337–345, 1992.

[217] Xi He, Ashwin Machanavajjhala, and Bolin Ding. Blowfish privacy: tuning privacy-utility trade-offs using policies. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 1447–1458.

[218] Daniel Kifer and Ashwin Machanavajjhala. Pufferfish: A framework for mathematical privacy definitions. In *ACM Transactions on Database Systems*, volume 39, page 3, 2014.

[219] B. A. Beckwith, R. Mahaadevan, U. J. Balis, and F. Kuo. Development and evaluation of an open source software tool for deidentification of pathology reports. *BMC Medical Informatics and Decision Making*, 6:12, 2006.

[220] Latanya Sweeney. Replacing personally-identifying information in medical records, the scrub system. In *Proceedings of the AMIA Annual Fall Symposium*, page 333. American Medical Informatics Association, 1996.

[221] Venkatesan T Chakaravarthy, Himanshu Gupta, Prasan Roy, and Mukesh K Mohania. Efficient techniques for document sanitization. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 843–852. ACM, 2008.

[222] Wei Jiang, Mummoorthy Murugesan, Chris Clifton, and Luo Si. t-plausibility: semantic preserving text sanitization. In *International Conference on Computational Science and Engineering*, volume 3, pages 68–75, 2009.

[223] G. Szarvas, R. Farkas, and R. Busa-Fekete. State-of-the-art anonymization of medical records using an iterative machine learning framework. *Journal of the American Medical Informatics Association*, 14(5):574–580, 2007.

[224] O. Uzuner, T. C. Sibanda, Y. Luo, and P. Szolovits. A de-identifier for medical discharge summaries. *Artificial Intelligence in Medicine*, 42(1):13–35, 2008.

[225] Zengjian Liu, Yangxin Chen, Buzhou Tang, Xiaolong Wang, Qingcai Chen, Haodi Li, Jingfeng Wang, Qiwen Deng, and Suisong Zhu. Automatic de-identification of electronic medical records using token-level and character-level conditional random fields. *Journal of Biomedical Informatics*, page in press, 2015.

[226] Azad Dehghan, Aleksandar Kovacevic, George Karystianis, John A. Keane, and Goran Nenadic. Combining knowledge- and data-driven methods for de-

identification of clinical narratives. *Journal of Biomedical Informatics*, page in press, 2015.

[227] Hui Yang and Jonathan Garibaldi. Automatic detection of protected health information from clinic narratives. *Journal of Biomedical Informatics*, page in press, 2015.

[228] Guido Zuccon, Daniel Kotzur, Anthony Nguyen, and Anton Bergheim. De-identification of health records using Anonym: Effectiveness and robustness across datasets. *Artificial Intelligence in Medicine*, 61:145–151, 2014.

[229] Mohammad Hossein Manshaei, Quanyan Zhu, Tansu Alpcan, Tamer Bacsar, and Jean-Pierre Hubaux. Game theory meets network security and privacy. *ACM Computing Surveys*, 45(3):25, 2013.

[230] Milind Tambe. *Security and Game Theory: Algorithms, Deployed Systems, and Lessons Learned*. Cambridge University Press, 2011.

[231] Wan Zhiyu, Yevgeniy Vorobeychik, Weiyi Xia, Ellen Wright Clayton, Murat Kantarcioglu, Ranjit Ganta, Raymond Heatherly, and Bradley Malin. A game theoretic framework for analyzing re-identification risk. *PLoS One*, page in press, 2015.

[232] Michael Brückner and Tobias Scheffer. Stackelberg games for adversarial prediction problems. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 547–555, 2011.

[233] Murat Kantarcioğlu, Bowei Xi, and Chris Clifton. Classifier evaluation and attribute selection against active adversaries. *Data Mining and Knowledge Discovery*, 22(1-2):291–335, 2011.

[234] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2011.

[235] Einat Minkov, Richard C Wang, and William W Cohen. Extracting personal names from email: applying named entity recognition to informal text. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 443–450. Association for Computational Linguistics, 2005.

[236] Son Doan and Hua Xu. Recognizing medication related entities in hospital discharge summaries using support vector machine. In *International Conference on Computational Linguistics: Posters*, pages 259–266. Association for Computational Linguistics, 2010.