Taming Evasions in Machine Learning Based Detection Pipelines

Alex Kantchelian

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-144
http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-144.html

August 12, 2016

Taming Evasions in Machine Learning Based Detection Pipelines

by

Alex Kantchelian

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Anthony D. Joseph, Co-chair
Professor J. D. Tygar, Co-chair
Professor Deborah Nolan

Summer 2016

Taming Evasions in Machine Learning Based Detection Pipelines

Abstract

Taming Evasions in Machine Learning Based Detection Pipelines

by

Alex Kantchelian

Doctor of Philosophy in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Anthony D. Joseph, Co-chair

Professor J. D. Tygar, Co-chair

This thesis presents and evaluates three mitigation techniques for evasion attacks against machine learning based detection pipelines. Machine learning based detection pipelines provide much of the security in modern computerized system. For instance, these pipelines are responsible for the detection of undesirable content on computing platforms and Internet-based services, such as malicious software and email spam. By its adversarial nature, the security application domain exhibits a permanent arms race between attackers who aim to avoid, or *evade*, detection and the pipeline's maintainers whose aim is to catch all undesirable content.

The first part of this thesis examines a defense technique for the concrete application domain of comment spam on social media. We propose *content complexity*, a compression-based normalized measure of textual redundancy that is mostly insensitive to the underlying language used and adversarial word spelling variations. We demonstrate on a real dataset of tens of millions of comments that content complexity alone achieves 15 percentage points higher precision than a state-of-the-art detection system.

The second part of this thesis takes a quantitative approach to evasion and introduces one machine learning algorithm and one learning framework for building hardened detection pipelines. Both techniques are generic and suitable for a large class of application domains. We propose the *convex polytope machine*, a non-linear large-scale learning algorithm which aims at finding a large-margin polytope separator and thereby decrease the effectiveness of evasion attacks. We show that as a general purpose machine learning algorithm, the convex polytope machine displays an outstanding trade-off between classification accuracy and computational efficiency. We also demonstrate on a benchmark handwritten digit recognition task that the convex polytope machine is quantitatively as evasion-resistant as a classic neural network.

We finally introduce *adversarial boosting*, a boosting-inspired framework for iteratively building ensemble classifiers that are hardened against evasion attacks. Adversarial boosting operates by repeatedly constructing evasion attacks and adding the corresponding corrective sub-classifiers to the ensemble. We implement this technique for decision tree sub-classifiers by constructing the first exact and approximate automatic evasion algorithms for tree ensembles. For our benchmark

task, the adversarially boosted tree ensemble is respectively five times and two times less evasion-susceptible than regular tree ensembles and the convex polytope machine.

To my grandmother

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgments

During my years at Berkeley, I have been fortunate to meet colleagues, friends and family who in one form or another left a lasting impression, and to whom I now express my deepest gratitude.

I would like to start by thanking my co-advisors, Professors Anthony D. Joseph and J. D. Tygar. Their constant support, conversation and trust has made this thesis possible, and I am further indebted for their insights on an effective technical communication style. I would also like to thank Professors Peter L. Bartlett, Richard M. Karp, Christos H. Papadimitriou, Vern Paxson and David Wagner for their inspiring intellectual rigor, inside and outside of the classroom.

I am also thankful to my colleagues Sadia Afroz, Rekha Bachwani, Riyaz Faizulabhoy, Michael C. Tschantz, Vaishaal Shankar, George Yiu and Tony Wu for their enthusiastic collaboration and invaluable conversations. I feel especially obliged to Justin Ma for his guidance in my initial year and to Ling Huang and Brad Miller for their continued engagement, both on research and personal grounds. I would also like to thank our program support assistant, Angie Abbatecola for her visible and invisible help.

I am deeply grateful to my parents Karmena and Leonid, my grandmother Rositza and my late grandparents for kindling a passion for science and engineering within me, and for their firm support and encouragement throughout my academic life.

I would like to thank Milad Odabaei for having made Berkeley feel home in his presence and for sympathetically enduring the unfolding of my research ideas over the past years and the writing of this thesis. In addition, I want to extend my deepest gratitude to friends and family who have filed my years in Berkeley with beautiful memories, including Golayoun, Jeanie, Shirin, Simon and Vandad as well as Michel Brun, Nicolas Eid, Antony Lee, Chloe Matte, Sofia Medina Ruiz and my quasi-roommate but complete-friend Denia Djokic. I am particularly grateful to my swimming partners and dear friends Christophe Cochet and Faiza Sefta whose company made my transition to UC Berkeley wonderful, and I wish I could always keep Sonia El Hedri and Emilia Esposito close to me.

The research presented in this thesis was supported by Intel's Science and Technology Center for Secure Computing Research for the User's Benefit (ISTC SCRUB), NSF grants 0424422 (TRUST) and 1139158, the Freedom 2 Connect Foundation, US State Dept. DRL, LBNL Award 7076018, DARPA XData Award FA8750-12-2-0331, and gifts from Amazon, Google, SAP, Apple, Cisco, Clearstory Data, Cloudera, Ericsson, Facebook, GameOnTalis, General Electric, Hortonworks, Huawei, Intel, Microsoft, NetApp, Oracle, Samsung, Splunk, VMware, WANdisco and Yahoo!. The opinions in this thesis are solely those of its author and do not necessarily reflect the opinions of any funding sponsor or the United States Government.

This thesis ties together work that in essence tackles the problem of evasion-resistant machine learning. The three approaches of Chapters 3, 4 and 5 correspond to the chronological progression of my research within SecML, my advisors' research group. These are in order the work on comment spam detection (Kantchelian et al. 2012), the convex polytope machine (Kantchelian et al. 2014) and adversarial tree ensembles (Kantchelian et al. 2016). As supplemental material to Chapter 4, the authors of (Hao et al. 2016) demonstrate the effectiveness of the convex polytope machine for the task of malicious URL detection. The SecML group has generally explored questions relevant to large-scale machine learning based detection pipelines, particularly in the context of malware detection and its associated labeling issues (Kantchelian et al. 2013; Miller et al. 2014; Kantchelian et al. 2015; Miller et al. 2016).

Chapter 1

Introduction

This thesis focuses on the case where machine learning is used for driving security critical decisions. Without automatic undesirable content detection, many Internet-based communication services, from regular email to the most famous social networks, would suffer reduced utility to the end user as the ratio of useful versus useless content decreases. If one can argue that machine learning simply improves the end user experience in this case, there are also many security applications with potentially catastrophic consequences. Consider a cyber-physical installation such as a nuclear power plant using machine learning to prevent execution of malicious code on its control systems, or, closer to the end-user, self-driving cars which extensively use machine learning to navigate their environment.

Present day computing systems produce data at phenomenal rates, both in the form of user generated content or general cyber and physical measurements. On one hand, data storage, computer networking and distributed computing together provide the underlying means for the persistent representation of those measurements. On the other hand, machine learning essentially turns this raw and otherwise inert data into algorithms which accomplish some useful tasks. The subfield of adversarial machine learning (Barreno et al. 2006) examines the security properties of machine learning in adversarial environments. Broadly speaking, the environment of a machine learning-based system is said to be adversarial when the goals of the system's designers and the environment's agents are at least partially conflicting. Adversarial agents exist across a large spectrum of targeted systems, internal goals and agent capabilities. For instance, spammers target Internet-based communication services for monetary gain by typically sending product advertisements (Levchenko et al. 2011; McCoy et al. 2012) and financial scams (Isacenkova et al. 2014) to end users. Their means include highly automated account creation and message posting pipelines which often rely on a distributed botnet infrastructure. At the other end of the spectrum, state sponsored attacks targeting critical physical infrastructure are becoming commonplace. Those attackers can count on considerable technical and financial resources and are often highly trained individuals.

In adversarial environments, the machine learning component of the targeted systems constitutes and additional part available for attackers to subvert. In other words, the inclusion of machine learning in a security critical system augments the attack surface (Howard and Lipner 2009) available to the system's adversaries. Somewhat paradoxically, this increase in attack surface happens *despite* the fact that machine learning is frequently deployed for protecting the pre-existing system from adversaries. This conceptual flaw can be readily understood as a mismatch between the goals of the machine learning component and the system's defender. Starting with an existing system S we want to protect using some machine learning component M, the security practitioner would today proceed to use M to protect S. However, this does not accomplish the defender's goals because the resulting system is no longer S alone, but a composition of S and M, which we loosely denote by $S \cup M$. Hence, the conceptually proper approach is to use M to protect $S \cup M$ instead. The tricky self-referential nature of the correct approach is the implicit but fundamental reason why the simpler approach of using M to protect S is systematically applied.

To understand the nature of the additional attack surface presented by $S \cup M$, it is useful to consider the *learning* and *inference* phase of machine learning separately. In the learning phase, we transform some training data into an algorithm which purpose is to protect the system by performing some detection of dangerous environment inputs for S. This training data will usually contain examples of both benign and dangerous inputs so that the resulting algorithm is effectively taught by example. In practice, this algorithm is a parametrized instantiation of an abstract structural algorithm. The set of parameters is traditionally called the *model* and the structural algorithm it parametrizes is called the *model class*. In the inference phase, we simply apply the previously learned algorithm to the inputs.

Chapter 2 will paint a finer picture of the type of attacks to which machine learning is susceptible. From a higher level perspective, an attacker can choose to target either one of the learning or inference phases. Attacks targeting the learning phase are said to be *poisoning* because they essentially amount to tampering with part of the training data to modify the learned model to the attacker's advantage. Generally speaking, a poisoning attack aims at decreasing the model's accuracy by increasing the number of times the model confuses dangerous inputs for benign ones and vice versa. Note that in realistic systems, it is typically easy for attackers to gain partial write access to the training set. Indeed, deployed systems need to regularly update their detection models by training on fresh data which in turn is partially controlled by the adversary. By definition, the adversary creates the malicious inputs, and depending on the volume of legitimate inputs naturally present, the adversary might additionally be able to control a non-negligible portion of those benign inputs by crafting his own.

The second family of attacks against machine learning takes place during the inference phase. Generally speaking, attacking the inference phase means finding a *surprising* input, that is, an input which the machine learning model and a human expert would label in a radically different manner. An interesting example is voice command spoofing (Carlini et al. 2016): one can construct audio sequences which on one hand do not sound like anything recognizable and on the other hand are happily interpreted as voice commands by devices with such capabilities. This surprising behavior is highly undesirable: an attacker can covertly direct the victim's phone to perform unwanted actions such as taking pictures or sending text messages in full presence of the victim.

This thesis focuses on a type of inference phase attacks called *evasions*. An evading attacker aims at evading detection from the model. For the attacker, this amounts to crafting a dangerous input which the model mistakenly recognizes as a benign input. At this point of the exposition, we need to look into the structure of a machine learning based detection pipeline to understand how an attacker might achieve evasion.

Figure 1.1 presents a high level description of a machine learning based detection pipeline during the inference phase. Such a pipeline can be understood a function that maps an observation into a decision. The type of the observation is specific to the application domain. For instance, these could be emails, executable files or network traces; in which case the corresponding decisions would be spam/ham message, malicious/benign executable and attack/legitimate traffic respectively. Because virtually all machine learning models operate on real-valued vectors, the observation is first embedded into vectorial space during the feature extraction step. Each dimension of the space represents a salient feature of the observation. Unlike the machine learning model which can be pretty generic, the feature extractor is highly dependent on the specific application domain.

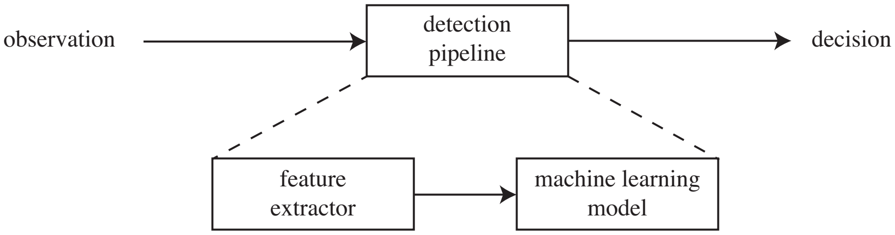


Figure 1.1: Simplified anatomy of a machine learning based detection pipeline for the inference phase. The feature extractor "adapts" the input into a format machine learning understands.

As can be seen from the decomposition of the detection pipeline, an attacker can achieve evasion in at least two ways. When crafting its evading observation, the attacker can choose to target either the feature extractor or the machine learning model itself. Evasion attacks that target the feature extraction step take advantage of the loss of information between the original observation and its vectorial embedding. Evasion attacks that target the machine learning model take advantage of the imperfect nature of machine learning. This decomposition motivates the two parts of this thesis: evasion attacks that target the feature extraction step, and evasion attacks that target machine learning itself.

Because the feature extraction step is domain specific, we ground Chapter 3 in the practical application of comment spam detection on a social-media platform. We introduce a new metric, *content complexity*, that is robust to widespread evasion technique based on spelling variations. We leverage this metric to create a small set of features well-adjusted to comment spam detection by computing the content complexity over groupings of messages sharing the same author, the same sender IP, the same included links, etc. We evaluate the new features against an exact set of tens of millions of comments collected over a four months period and containing a variety of websites, including blogs and news sites. The data was provided to us with an initial spam labeling from an industry competitive source. Nevertheless the initial spam labeling had unknown performance characteristics. To train a logistic regression on this dataset using our features, we derive a simple mislabeling tolerant logistic regression algorithm based on expectation-maximization, which we show generally outperforms the plain version in precision-recall space. We show that our method can operate at an arbitrary high precision level, and that it significantly dominates, both in terms of precision and recall, the original labeling, despite being trained on it alone.

In the second part of this thesis, we take a closer look at the machine learning itself and abstract the feature extraction part to the highest possible extent. In contrast to Chapter 3, this allows us to simulate powerful evading adversaries that operate in the embedding vectorial space and to make general evasion robustness claims across a wild spectrum of machine learning algorithms.

In particular, Chapter 4 introduces the *Convex Polytope Machine* (CPM), a large-margin based, non-linear algorithm for fast large scale model learning. Because of the particular model structure, we are able to study in an exact quantitative fashion its susceptibility to evasions. We demonstrate that on one hand the CPM exhibits outstanding computational performance and competitive accuracy while on the other hand being susceptible to evasions. We also demonstrate particularly effective evasion attacks against tree ensembles in Chapter 5, a class of non-linear, non-differentiable models which contains the popular gradient boosted trees and random forests model classes. For a consequent selection of evasion attack types, Chapter 6 shows that tree ensemble models systematically rank at the bottom in terms of robustness, notwithstanding their competitive accuracy.

To end on a positive note, Chapter 6 also evaluates *adversarial boosting*, our novel tree ensemble building algorithm based on boosting which creates significantly harder to evade models without sacrificing the detection accuracy. As a way to harden otherwise weak machine learning models, adversarial boosting offers interesting perspectives to the security decision pipelines designers and practitioners.

Chapter 2

Background and Prior Work

This chapter formalizes and contextualizes the concepts presented earlier. We start with a succinct overview of machine learning in the context of security applications. We then present a well known taxonomy of attacks targeting machine learning and finally focus on evasion attacks for which we provide a literature review of examples and suggested counter-measures.

2.1 Machine Learning for Security Applications

Broadly speaking, machine learning aims at turning data that is exemplary of a task to accomplish into an algorithm which accomplishes the said task. Machine learning has a plethora of applications. In natural language processing, machine learning can perform automatic translation (Sutskever et al. 2014) and speech-to-text transcription (Graves et al. 2013); in robotics, it enables environment navigation and interaction through obstacle detection (Benenson et al. 2012) and movement planning (Peters and Schaal 2008); in artificial intelligence, machine learning has been the driving force behind the fast progress of automatic picture description systems (Vinyals et al. 2015).

In this thesis, we focus on detection tasks, or yes/no decision tasks. For such tasks, our aim is to train the machine to answer binary yes/no questions. In the context of computer security, some examples of binary questions of interest are:

- Is this email spam?
- Is this executable file malicious?
- Is this network traffic trace indicative of an attack?

Formally, let Ω be an *observation space* that is application domain specific, $\omega \in \Omega$ an observation and $\mathcal{Y}$ a set of two elements. For the three previous examples, Ω would be respectively defined as the set of all possible emails, executable files and network traffic traces. We choose the two specific elements of $\mathcal{Y}$ to lighten the notational formalism in each part of this thesis. Specifically, in Chapter 3, we take $\mathcal{Y} = \{0; 1\}$, and in every subsequent chapter $\mathcal{Y} = \{-1; 1\}$. Throughout this thesis, the positive element 1 always represents a malicious, dangerous, or otherwise undesirable observation, and the non-positive element stands for a benign observation.

A machine learning based detection pipeline provides a mapping Φ between Ω and $\mathcal{Y}$:

$$\begin{aligned} \Phi : \Omega &\rightarrow \mathcal{Y} \\ \omega &\mapsto \Phi(\omega) \end{aligned}$$

Machine learning provides a generic approach to building such mappings. Instead of accepting arbitrary inputs from Ω, a machine learning classifier operates in a real-valued vectorial space. Let $d \in \mathbb{N}^*$ be the number of dimensions of the input space. A classifier c is a mapping between *instances* $x \in \mathbb{R}^d$ and *labels* $y \in \mathcal{Y}$. For all practical classifiers, c is constructed from a thresholded real-valued classifier f:

$$\begin{aligned} f : \mathbb{R}^d &\rightarrow \mathbb{R} \\ x &\mapsto f(x) \end{aligned}$$

so that we have $c(x) = 1 \Leftrightarrow f(x) > \tau$ for some threshold value τ, most commonly fixed at $\tau = 0$. $f(x)$ is the *signed margin* of the binary classifier c and is classically understood as a measure of confidence for the decision. The larger $|f(x)|$ is, the more confident c is about the $c(x)$ decision. While c is technically our binary classifier, we more often work with the signed margin f directly.

Because machine learning only provides domain-agnostic tools, Φ operates in two phases. In the first phase, a *feature extraction* function $\psi : \Omega \rightarrow \mathbb{R}^d$ embeds the observation ω into a point $\psi(\omega) \in \mathbb{R}^d$. The embedding ψ is largely domain-dependent but generally aims at mapping the informative characteristics of the observation into dimensions of $\mathbb{R}^d$. In the second phase, the machine learning classifier c performs the inference $f(\psi(\omega))$. We refer to the dimensions of the vector $\psi(\omega)$ as the *feature* dimensions.

Machine learning is the theoretical and practical body of work which aims at constructing "good" classifiers f. Among all general purpose approaches for constructing f, *Empirical Risk Minimization* dominates by far current machine learning techniques, both in terms of practical results and theoretical understanding.

Empirical Risk Minimization

In this thesis, we restrict our interest to *supervised* machine learning. Given a *labeled* dataset $\mathcal{D}$ of n instances $x^{(1)}, \ldots, x^{(n)} \in \mathbb{R}^d$ with respective labels $y_1, \ldots, y_n \in \{-1; 1\}$ and a *loss* function $\ell : \mathbb{R} \rightarrow \mathbb{R}_+$, the *empirical risk* of classifier f over dataset $\mathcal{D}$ is defined as:

$$R_{\mathcal{D}}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i f(x^{(i)}))$$

In this thesis, i is the preferential variable name for indexing the dataset instances, and we use it along with the superscript parenthesized notation.

For example, $R_{\mathcal{D}}(f)$ counts the proportion of misclassification errors made by f when ℓ is taken to be the 0/1 loss:

$$\forall q \in \mathbb{R}, \ell_{0/1}(q) = \begin{cases} 0 \text{ if } q > 0 \\ 1 \text{ if } q \leq 0 \end{cases}$$

A supervised learning algorithm aims at finding a classifier f such that:

1. f has a low empirical risk on the given training set $\mathcal{D}$, and
2. f has a low empirical risk on future, not yet seen, datasets.

Property (2) is often referred to as the ability of f to *generalize* well given its initial training dataset $\mathcal{D}$. Obviously, the training dataset and future unseen datasets must share some common property for such endeavor to be meaningful. In the theoretical analysis of machine learning algorithms, we formalize this requirement by introducing an underlying but unknown probability distribution which generates an infinite amount of independent and identically distributed labeled instances (x, y), so that $\mathcal{D}$ and all subsequent datasets emanate from the same stochastic process. In practice, this is almost never the setting in which machine learning is applied. The main issue is the presence of time. In reality, the (x, y) pairs many neither be independent nor identically distributed. The theoretical framework is nonetheless important for establishing necessary conditions for good machine learning algorithms: if an algorithm fails to satisfy either (1) or (2) in the framework, then we have no hope for it to be effective in practice. Algorithms that satisfy (1) and (2) under independence and identical distribution of instances are said to be *consistent*.

Empirical Risk Minimization (ERM) is a generic approach to constructing consistent learning algorithms. Let $\mathcal{F}$ be a *model class*, that is, a set of candidate classifiers:

$$\mathcal{F} \subset \left\{ f \,\middle|\, f : \mathbb{R}^d \to \mathbb{R} \right\}$$

As its name suggests, the ERM approach finds a classifier $f \in \mathcal{F}$ which minimizes the empirical risk on the dataset $\mathcal{D}$:

$$\min_{f \in \mathcal{F}} R_{\mathcal{D}}(f) \tag{2.1}$$

By definition, ERM itself focuses on satisfying condition (1). An important body of theoretical work shows that for the proper choice of model class $\mathcal{F}$, the solution of problem (2.1) also satisfies condition (2). The relationship between $\mathcal{F}$ and generalization power takes the form of an intuitive Occam's razor argument: the less complex, or the smaller the model class is, the better the chosen classifier f generalizes. Hence, there is a trade-off between conditions (1) and (2) as the simpler $\mathcal{F}$ is, the less likely we are to find a good classifier describing $\mathcal{D}$. This dilemma is often referred to as *underfitting* v.s. *overfitting*, or a *bias-variance trade-off*.

Several measures of complexity and corresponding generalization bounds have been developed. Historically important, the first such measure is the *Vapnik-Chervonenkis* (VC) *dimension* (Vapnik 1995), or shattering dimension. The VC dimension of $\mathcal{F}$ is the maximal size of a dataset of labeled instances $(x^{(i)}, y_i) \in \mathcal{R}^d \times \{-1; 1\}$ for which there exists a classifier $f \in \mathcal{F}$ with empirical

0/1-loss being 0. An alternative, finer-grained measure of model class size is the *Rademacher complexity* (Bartlett and Mendelson 2003). We will use this latter measure when stating our generalization bounds for the Convex Polytope Machine in Chapter 4.

We now give a brief overview of common model classes $\mathcal{F}$, starting with linear models.

Linear Classifiers

f is a linear classifier if and only if there exists a vector $w \in \mathbb{R}^d$ and a term $b \in \mathbb{R}$ such that for all $x \in \mathbb{R}^d$ we have

$$f(x) = \sum_{j=1}^{d} x_j w_j + b = x^\intercal w + b$$

where for vectors $u, v \in \mathbb{R}^d$, the subscript notation u_j denotes the value of dimension j of u and $u^\intercal v$ is the dot, scalar or inner product and j is the preferential variable name indexing the feature dimensions of x. Vector w is called the weight vector because it multiplicatively weights each dimension of x, and the constant b is called the bias or intercept term. For the sake of notational brevity, when working with linear classifiers, we always include the bias term in the weight vector w by appending a constant unitary dimension to all instances x. That is, ψ constructs instances x such that $x_d = 1$, and w_d plays the role of b.

Owing to their long history and relatively simple structure, linear classifiers are by and large well understood. In particular, there exists a profusion of algorithms for finding a good weight vector w given some training data: the perceptron algorithm (Freund and Schapire 1999), the naive Bayes algorithm (Russell and Norvig 2009), logistic regression (McCullagh and Nelder 1989), linear support vector machine (SVM) (Cortes and Vapnik 1995), confidence weighted linear classification (Dredze et al. 2008), etc.

The perceptron, logistic regression and linear SVM models belong to the ERM family of algorithms. In particular, logistic regression minimizes the logistic loss ℓ_{LR} which is a smooth, convex upper-bound of the 0/1 loss:

$$\ell_{LR}(q) = \log_2 (1 + \exp(-q))$$

It is common to constrain the model class complexity using the L_1 or L_2-norm of w, in which case we talk about L_1 or L_2-regularized logistic regression respectively. That is, for a given positive constant $\Lambda \in \mathbb{R}_+$ and norm $\rho \in \{1; 2\}$, the L_ρ-regularized logistic regression model class is:

$$\mathcal{F}_{\rho,\Lambda} = \left\{ f \;\middle|\; \exists w \in \mathbb{R}^d, \|w\|_\rho \leq \Lambda;\; \forall x \in \mathbb{R}^d, f(x) = x^\intercal w \right\}$$

Generally speaking, L_1 regularization results in a sparse weight vector solution for the learning problem (2.1), where only a few dimensions of w are non-zero. The L_2-regularized logistic regression distributes weights more evenly across the dimensions of w. On the implementation side, we note that because ℓ_{LR} is a convex function and the set of admissible w is convex as well, the machinery of convex optimization (Boyd and Vandenberghe 2004) is often successfully invoked for solving the minimization problem (2.1), even for datasets of millions of samples n and feature dimensions d (Fan et al. 2008).

Neural Networks

The neural network model class is obtained by layered composition of linear models and non-linear functions sometimes called *activation* functions (LeCun et al. 2012). Formally, let $\sigma : \mathbb{R} \to \mathbb{R}$ be a non-linear function. For neural networks, σ is commonly chosen as a sigmoidal function. Informally, σ is sigmoidal if the following conditions hold:

1. $\sigma(0) = 0$ (centered)
2. $\lim_{q \to +\infty} \sigma(q) = 1$ (unitary asymptote)
3. σ is non-decreasing
4. $\sigma(q) = -\sigma(-q)$ (symmetry)

Typically used sigmoidal functions are:

$$\begin{aligned}\sigma(q) &= \tanh(q),\\ \sigma(q) &= \frac{q}{\sqrt{1+q^2}}, \text{ and}\\ \sigma(q) &= \frac{2}{\pi} \arctan\left(\frac{\pi}{2} q\right)\end{aligned}$$

Let $p \geq 1$ be a natural number. f belongs to the model class of p-layer deep neural networks when there exists $l_1, \ldots, l_{p-1} \in \mathbb{N}^*$ (the number of *units*, or *artificial neurons* per internal layer), weight matrices $W_1 \in \mathbb{R}^{l_1 \times d}, W_2 \in \mathbb{R}^{l_2 \times l_1}, \ldots, W_p \in \mathbb{R}^{1 \times l_{p-1}}$ and offset vectors $b_1 \in \mathbb{R}^{l_1}, b_2 \in \mathbb{R}^{l_2}, \ldots, b_p \in \mathbb{R}$ such that f can be written as:

$$\forall x \in \mathbb{R}^d,\ f(x) = W_p \sigma\left(W_{p-1} \sigma\left(\ldots \sigma\left(W_1 x + b_1\right) \ldots\right) + b_{p-1}\right) + b_p$$

where Mu is the matrix-vector product between matrix M and vector u, and σ operates element-wise on its vector argument. For $p = 1$, the model class collapses to linear classifiers:

$$f(x) = W_1 x + b_1$$

where $W_1 \in \mathbb{R}^{1 \times d}$ and $b_1 \in \mathbb{R}$. For $p > 1$, we say the model has $p - 1$ hidden layers. Neural networks are considerably more expressive than linear classifiers: any reasonable function can be approximated by a deep enough neural network (Hornik 1991).

ERM is the predominant method for finding the coefficients of $W_1, \ldots, W_p$ and $b_1, \ldots, b_p$. Unlike logistic regression, the optimization problem (2.1) is generally non-convex, even when using a convex loss function such as ℓ_{LR}. Stochastic Gradient Descent (SGD) is the method of choice for finding good enough solutions (Bottou 2012). Historically, SGD is often called *back-propagation*, because of the form ∇f takes from the chain rule for gradients of function compositions.

Recent work has considerably expanded the collection of tricks available for training neural networks. For instance, a simple rectifier function $\sigma(q) = \max(0, q)$ can be as effective as more complex sigmoids (Glorot et al. 2011). Pre-training (Erhan et al. 2010) and drop-out (Srivastava et al. 2014) are also novel regularization techniques for neural networks.

Kernel Methods

Kernel methods rely on a pre-specified similarity function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ to perform prediction. f is a kernel classifier when there exists a weight vector $w \in \mathbb{R}^n$ and points $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}^d$ commonly referred to as *support vectors*, such that f can be written as:

$$f(x) = \sum_{i=1}^{n} w_i K(x, x^{(i)})$$

Popular examples of similarity functions K are the linear kernel $K(u, v) = u^\intercal v$, the polynomial kernel of degree $t \in \mathbb{N}^*$ and constant term $b \in \mathbb{R}$, $K(u, v) = (u^\intercal v + b)^t$ and the Radial Basis Function (RBF), also called Gaussian kernel $K(u, v) = \exp\left(-\gamma \|u - v\|^2\right)$ with a positive real-valued parameter γ. Note that for the linear kernel choice, the model class collapses to linear classifiers again.

Kernel methods have enjoyed a sustained popularity because of the flexibility for the choice of K. That is, for problems where there exists a good notion of similarity between instances, a kernel classifier with an adapted choice of K can achieve excellent performance.

In practice, the most popular incarnation of kernel methods is the kernel Support Vector Machine (SVM). Kernel SVM is another example of the ERM line of algorithms. The loss function for SVM ℓ_H is a convex but non-smooth upper-bound of the 0/1 loss and is referred to as the hinge loss because of its shape:

$$\forall q \in \mathbb{R}, \ell_H(q) = \max(0, 1 - q)$$

One can also prove that under ERM, and under certain conditions on the kernel K, the optimal choice of support vectors for SVM is simply the set of original training instances $(x^{(i)})$. In general, SVMs have a deep connection with convex optimization concepts and a rich theory surrounding those exist (Boser et al. 1992; Vapnik 1995; Thomas Hofmann 2008).

Decision and Regression Trees

A regression tree T is a binary tree where each internal node NODE $\in T$.NODES is labeled with a logical predicate NODE.PREDICATE over the dimensions of x, the two children of NODE are by convention labeled NODE.TRUE and NODE.FALSE and finally each leaf LEAF $\in T$.LEAVES holds a real value LEAF.PREDICTION. For a given instance x, the prediction path in T is the path from the tree root T.ROOT to a leaf such that for each internal node NODE in the path, NODE.TRUE is also in the path if and only if NODE.PREDICATE holds over the concrete instance x. The model prediction is the leaf value of the prediction path. Figure 2.1 presents an example of a regression tree.

Node predicates can be arbitrary complex. In this thesis, we focus on the most prevalent type of predicates encountered in practice: inequalities involving a single feature dimension.

Under ERM, we would have to find the best possible tree structure, predicates and leaf values with respect to a given loss function and the training dataset. In practice, this optimization problem is known to be computationally intractable. Instead, the tree classifier is greedily grown from the root by finding at every step the predicate that would separate best the two classes, and thus

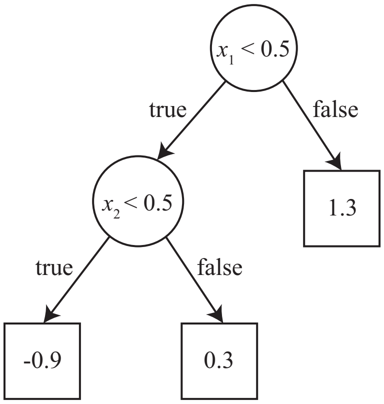


Figure 2.1: Example of regression tree. For a given instance, inference is done by traversing the tree from the root to a leaf such that each branch choice is consistent with the valuation of the corresponding node predicate. For example, for instance $x = (1, 0)$, the prediction of the model is 1.3.

give the "purest" leaves. The growing procedure is recursively applied until the current leaves are pure, or a stopping criterion is met, e.g., a maximal depth has been reached (Hastie et al. 2005). To control complexity and improve generalization, there also exists several tree simplification, or pruning techniques (Mehta et al. 1995; Kearns and Mansour 1998). In practice, decision trees are rarely used as classifiers on their own because of their relatively poor generalization performance when compared to competing approaches.

Tree Ensembles

Informally, while a single decision tree is a *weak*[1] classifier, many weak classifiers taken together can form an accurate classifier. Specifically, summing together regression tree outputs is an effective way to improve the generalization performance. f is a sum-ensemble of trees if and only if there exists $K \geq 1$ regression trees $f_1, \ldots, f_K$ such that:

$$f(x) = \sum_{k=1}^{K} f_k(x)$$

The individual classifiers f_k are often referred to as *base* classifiers in the context of ensembles. There are two common approaches for building such ensembles: random forests and boosting. The *random forest* technique (Breiman 2001) is an indirect ERM procedure. In a random forest, each regression tree is built independently under ERM, with the added constraint that only predicates over a randomly selected subset of the feature dimensions are allowed.

Unlike random forests, *boosting* (Freund et al. 1999; Friedman 2001) has a direct ERM interpretation. Boosting is a greedy iterative approach where at each round, we find the best possible

[1] The *weak* terminology has a precise mathematical definition, see (Freund and Schapire 1995).

classifier to add to the ensemble. Formally, let $\mathcal{F}$ be a subset of the regression tree model class. The f_i are iteratively obtained as correction terms to the current model as follows:

$$\begin{aligned}
f_1 &= \operatorname*{argmin}_{h \in \mathcal{F}} R_{\mathfrak{D}}\left(h\right) \\
f_2 &= \operatorname*{argmin}_{h \in \mathcal{F}} R_{\mathfrak{D}}\left(f_1 + h\right) \\
&\dots \\
f_K &= \operatorname*{argmin}_{h \in \mathcal{F}} R_{\mathfrak{D}}\left(\sum_{k=1}^{K-1} f_k + h\right)
\end{aligned}$$

AdaBoost, or additive boosting is historically the first example of boosting (Freund and Schapire 1995). AdaBoost can be obtained from the above greedy ERM procedure by using the exponential loss (Friedman 2001):

$$\forall q \in \mathbb{R},\ \ell_{EXP}(q) = \exp(-q)$$

In practice, the logistic loss produces better results in an algorithm called *logitboost* (Friedman et al. 2000; Li 2012). Careful adaptations of logitboost for regression trees exist (T. Chen and Guestrin 2016) and have shown impressive results in numerous machine learning competitions.

Linear classifiers, common neural networks and kernel methods are all *differentiable* models and their gradient ∇f is often extremely useful both for learning and explanatory purposes. Decision trees and tree ensembles on the other hand are *piecewise constant* models for which differentiation is a useless tool.

2.2 Attacking Machine-Learning-based Detection Systems

When used in an adversarial setting, machine learning-based detection pipelines face a family of challenges. On one hand, the system designer uses general purpose machine learning techniques such as ERM to build the detection pipeline Φ. On the other hand, adversaries are by definition aiming at undermining the system, by any sensible means. Even under the generous assumption that the implementation of Φ itself is absolutely free of bugs, there still remains a consequent and potentially exploitable attack surface.

Taxonomy of Attacks

Every attack aims at increasing the error rate of Φ. Table 2.1 describes the four types of events that can happen for any given observation $\omega \in \Omega$. Attacks aim at either increasing the false positives or false negatives.

The authors of (Barreno et al. 2006) lay out the landscape of attacks Φ is susceptible to. In this taxonomy, an attack can be characterized along three dimensions: its influence capability, its specificity and the type of security violation it causes. Table 2.2 presents the possible characteristics for each dimension of the taxonomy. We succinctly expand on each attack dimension.

	benign ω	malicious ω
$\Phi(\omega) \leq 0$	True Negative	False Negative
$\Phi(\omega) > 0$	False Positive	True Positive

Table 2.1: Terminology for the decision events as a function of the ground truth nature of ω and the actual decision $\Phi(\omega)$.

Dimension	Type for dimension	
Influence	Causative Attacker can partially manipulate training set $\mathcal{D}$.	Exploratory Attacker can partially inspect the internals of Φ.
Specificity	Targeted Attacker is only interested in a specific type of observation ω.	Indiscriminate Attacker is indifferent to the type of observation ω.
Security Violation	Integrity Attacker increases false negatives.	Availability Attacker increases any type of error and renders Φ useless.

Table 2.2: Typology of possible attacks. Each one of the three rows is an independent descriptive axis.

The influence axis captures the amount of access to the system our attacker has. In causative, or training-time attacks, the attacker can effectively control a portion of the training dataset. This situation often occurs in practice, as by definition malicious observations are crafted by the attacker. This problem is exacerbated in online and active learning (Miller et al. 2014) settings. In exploratory, or testing-time attacks, the attacker can not modify Φ but has a given amount of knowledge over Φ and probing capacity in the form of crafting queries ω and receiving $\Phi(\omega)$ answers.

The specificity axis captures the focus of the attacker. In targeted attacks, the attacker only cares about causing a misclassification on a small subset of observations ω. In indiscriminate attacks, the specific nature of ω does not matter as much as the total number of misclassifications the attacker wants to cause.

Finally, the security violation axis captures the type of misclassification errors caused. In integrity attacks, the attacker increases the false negative errors, and thus *evades* detection. In availability attacks, the attacker aims at increasing false positive, false negative or both errors to a level where the prediction of Φ can no longer be acted on, rendering the system effectively useless.

This thesis focuses on exploratory attacks targeting the integrity of the detection system, that is, on testing-time evasion attacks, or simply evasions.

Evasions

Figure 2.2 presents the main ingredients of an evasion attack. We say that ω' is an evading observation for a given detection pipeline Φ if $\Phi(\omega') \leq 0$ and there exists a semantically-equivalent observation ω such that $\Phi(\omega) > 0$. In other words, ω' is a false negative and there exists a similar instance which Φ correctly classifies as malicious. This definition captures the adversarial process: if an attacker's nefarious input ω is correctly detected by the pipeline, then the attacker seeks to craft a new input ω' which both achieve the attacker's goals and is misclassified as benign.

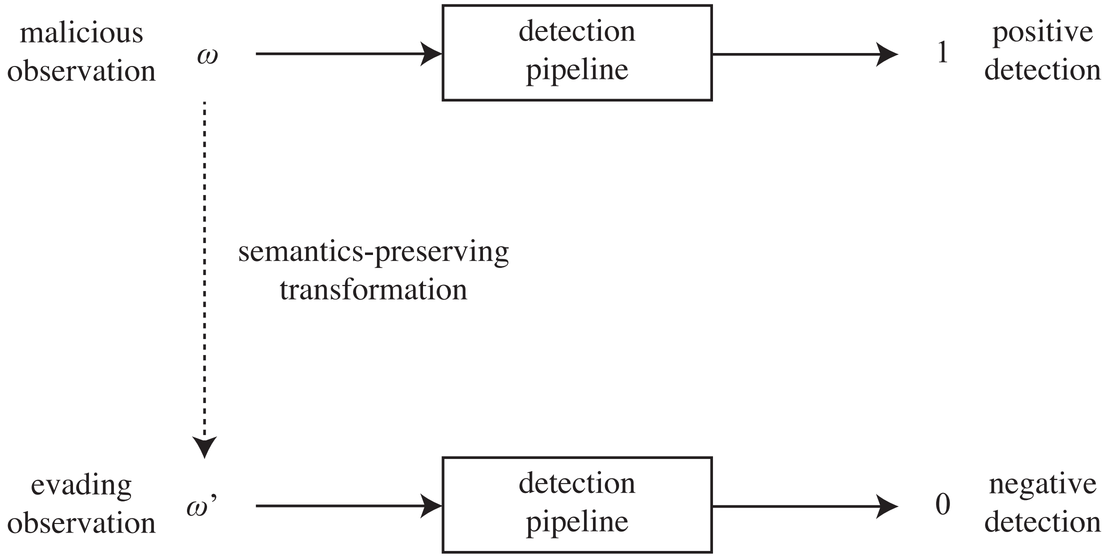


Figure 2.2: Definition of evasion.

Fully modeling the semantics-preserving transformation is, for most application domains, an extremely difficult task. However, for many domains of interest, one can capture important aspects of the transformation and study its effects on the detection pipeline. For example, in spam detection, one can more or less significantly misspell content words and still preserve the general meaning of the message. Alternatively, one can also replace the raw textual content altogether with a picture of it, for no loss of meaning to a human reader.

Quantifying the Evasion Susceptibility of Classifiers

We now introduce a simple but powerful formalization of the semantics-preserving concept which allows us to quantitatively study the evasion robustness properties of virtually all machine learning classifiers f. Here, we forgo for a moment the application-specific feature extraction phase ψ and directly work in the embedding space $\mathbb{R}^d$ where f operates.

We follow the definition of (Biggio et al. 2013). Let c be a classifier. For a given instance x and a given "distance" function $\Delta : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$, the optimal evasion problem is defined as:

$$\underset{x' \in \mathbb{R}^d}{\text{minimize}}\ \Delta(x, x') \quad \text{subject to } c(x) \neq c(x') \tag{2.2}$$

Setting the classifier c aside, the distance function Δ fully specifies (2.2), hence we later talk about d-evading instances, or d-robustness. In fact, for many practical complete detection pipelines Φ, we can exactly study evasion-related questions directly under formulation (2.2), with a judicious choice for d. That is, d can both be used to:

1. model the semantic distortion between instances x and x', or from a different perspective, model the cost the attacker has to pay for changing her initial instance x into x', and
2. model the effect of the feature extractor ψ.

In this thesis, we proceed as if the attacker cost is decomposable over the feature dimensions. In particular, part II presents results for four representative distances. We briefly describe those and their typical effects on the solution of (2.2).

The L_0 distance $\sum_{k=1}^{d} \mathbb{I}_{x_k \neq x'_k}$, or Hamming distance encourages the sparsest, most localized deformations with arbitrary magnitude. The techniques presented in this paper can also accommodate the case of non-uniform costs over features. This situation corresponds to minimizing $\sum_{k=1}^{d} s_k \mathbb{I}_{x_k \neq x'_k}$ where s_k are non-negative weights.

The L_1 distance $\sum_{k=1}^{d} |x_k - x'_k|$ encourages localized deformations and additionally controls for their magnitude.

The L_2 distance $\sqrt{\sum_{k=1}^{d} (x_k - x'_k)^2}$ encourages less localized but small deformations.

The L_∞ distance $\max_k |x_k - x'_k|$ encourages uniformly spread deformations with the smallest possible magnitude.

Note that for binary-valued embedding domains $x \in \{0; 1\}^d$, L_1 and L_2 reduce to L_0 and L_∞ results in the trivial solution value one for problem (2.2).

The feature extraction ψ can also be modeled in the optimal evasion problem by properly defining d and potentially complex inter-feature dependencies can be modeled. For instance, consider the common case of two mutually exclusive binary feature dimensions k and l such that for any observation ω, features $\psi(\omega)_k$ and $\psi(\omega)_l$ can never be both non-zero at the same time. This can be modeled in d as an additive term of the form $+B\mathbb{I}_{x_k \neq 0}\mathbb{I}_{x_l \neq 0}$ with a large-enough constant penalty value B. The approaches we present in Chapters 4 and 5 can readily accommodate such exclusion constraints directly, without introducing extra constants like B.

Evasion Attacks

Table 2.3 presents and organize prior work on designing attacks against detection pipelines, that is, methods that find for a given malicious observation ω a corresponding evading observation ω'. We categorize attacks across five dimensions: the considered application domain, the considered model class(es), the amount of knowledge the attacker has over the detection pipeline, the completeness of the attack, and its computational cost.

We distinguish three specific application domains: email spam, PDF malware and image recognition. In email spam, attack algorithms seek to minimally transform the content of a spam email to evade detection. For all prior works, these attacks consist in either adding words that are indicative of, or correlated with, benign messages, or replacing words associated to spam with more neutral synonyms. For malicious PDF detection, structural metadata replaces words. Features of interest are for example whether the PDF contains executable JavaScript code, the number of distinct fonts used and the number of objects the document contains. A semantics-preserving transformation would for example be allowed to increase the counts of fonts and objects, but not remove JavaScript code. For image recognition, the semantics of pictures is preserved under a large variety of transformations. For instance, arbitrary large modifications of a small number of pixels or very small perturbations over all pixels almost never affect our perception of an image. We also distinguish a generic application domain, where the specifics of the feature extraction ψ are abstracted away and the algorithms directly operate in the embedding space $\mathbb{R}^d$ and focus on solving problem (2.2).

The considered model classes in prior work are linear classifiers, kernel methods, neural networks and generally differentiable classes (diff.), tree ensembles (tree ens.) and convex decision boundary-inducting classifiers (Nelson et al. 2012), which we denote by linear+ as no off-the-shelf model class but linear classifiers is a proper subset of this category. In Chapter 4, we also introduce and study the Convex Polytope Machine, a model that is constructed by ensembling linear classifiers with the max operator (max ens.) instead of the addition operator as in boosting.

The amount of knowledge available over Φ largely informs the type of attack an adversary can carry out. We distinguish three categories. Φ might be fully available to the attacker (full) in the form of knowledge of the classifier parameters and the features used. Alternatively, in the query setting (query), the attacker can only use Φ as a black-box oracle by submitting arbitrary queries. Some prior works also make general assumptions over the form of Φ, for example the type of features that ψ might produce and the model class of f and have some knowledge over the training data used, typically by allowing access to a subset of it. We call this setting indirect knowledge (indirect).

The completeness category describes whether the evasion algorithm is always able to find an evading instance provided one exists, or, depending on the context, find an evading instance with the minimal amount of deformation. Exact evasion algorithms always return an optimal evading instance with respect to some criterion. By contrast, heuristic algorithms are not guaranteed to find the optimal instance and, in many cases, are not even guaranteed to find any evading instance.

We also provide an approximate indication of the amount of work required for finding evading instances. On one extreme, linear classifiers with full knowledge of the weight vector are easy to evade (Dalvi et al. 2004). On the other extreme, practical applications where the model class is arbitrary complex and only querying knowledge is allowed have a large computational cost, even under heuristic evasion (W. Xu et al. 2016). In our table, low cost methods are those which running time are no worse than proportional to the size of the evaded model, medium cost methods are theoretically super-linear but are reported to typically compute an evading instance in the order of seconds, and high cost methods are methods which are theoretically super-linear and result in unpredictably long generation time. Owing to its genetic algorithm approach, (W. Xu et al. 2016) is the only work with a high computational cost in this table.

Prior work	Domain	Model	Knowledge	Complete	Cost
Dalvi et al. (2004)	generic, email spam	linear	full	yes	low
Lowd and Meek (2005b)	email spam	linear	indirect, query	no	medium
Lowd and Meek (2005a)	generic	linear	query	yes	medium
Smutz and Stavrou (2012)	PDF malware	tree ens.	indirect	no	low
Nelson et al. (2012)	generic	linear+	query	yes	medium
Biggio et al. (2013)	generic, image recognition	diff.	full	no	low
Szegedy et al. (2013)	generic, image recognition	diff.	full	no	medium
Goodfellow et al. (2015)	generic, image recognition	diff.	full	no	low
Srndic and Laskov (2014)	PDF malware	kernel, tree ens.	indirect	no	low
W. Xu et al. (2016)	PDF malware	any	query	no	high
This thesis, Section 4.5	generic, image recognition	max ens.	full	yes	medium
This thesis, Section 5.4	generic, image recognition	tree ens.	full	no	low
This thesis, Section 5.5	generic, image recognition	tree ens.	full	yes	medium

Table 2.3: Contrasting prior work on attacks with the work presented in this thesis (last three rows).

In this thesis, we present the first exact evasion algorithms against both max ensembles of linear classifiers in Section 4.5 and tree ensembles in Section 5.4, when the models are fully known.

Unlike virtually all prior work which uses gradient descent strategies on differentiable models, our method casts the evasion problem into an Integer Linear Problem which can in turn be solved by off-the-shelf optimization software. This allows us to compute exact lower-bounds for the classifiers' evasion susceptibility. The main advantage of this technique is that when those lower-bounds are high enough for a given classifier, we can formally rule out entire classes of evasion attacks before even fielding the classifier. We also present a novel, fast and precise heuristic evasion method for tree ensembles that we later use to make those ensembles more robust against evasions.

Evasion Defenses

The dual of finding evading instances is finding ways to robustify, or harden detection pipelines against evasion. Table 2.4 categorizes the prior work pertaining to evasion defenses under five categories: application domain, type of defense family, estimated practical effectiveness, impact on classification accuracy and finally computational cost.

In addition to the three application domains encountered in attacks, we tackle the hardening of a comment spam detection pipeline. Comment and email spam share the similar trait that both types of messages degrade the end-user's experience. Nevertheless, comment spam has unique characteristics. Message lengths are usually considerably shorter, they exhibit a higher amount of non-English content and are sometimes meant for search engine optimization, to name a few differences.

We distinguish seven types of defense strategies. An often encountered strategy is to design a novel model class (new model). The potential pitfall of this approach is that the new model class is oftentimes designed with a weak adversary in mind. That is, the adversarial assumptions are often such that the novel class is unknown to, or mistaken by the adversary. This assumption is too strong for any practical adversarial setting and can be thought as violation of Kerckhoffs' principle in security: the security of a system should not exclusively rely on secrecy (Shannon 1949; Kerckhoffs 1883). In contrast, we derive an optimal evasion algorithm against our new model, the Convex Polytope Machine in Chapter 4 and empirically study its susceptibility to evasion. Another defense strategy is to design the feature extraction step ψ such that only features that are both informative and hard to manipulate by an adversary are embedded (ψ-level). It is unfortunately often the case that removing features that are easily modified but nonetheless correlated with malicious observations decreases the system's general accuracy. The regularization technique (regularization) aims at further constraining the model class $\mathcal{F}$. For example, in a differentiable model f, to make f more stable around its future predictions and thus more resistant to evasive perturbations, it might be valuable to flatten-out the prediction landscape around training instances $x^{(i)}$. This can be achieved by constraining the size of $\nabla f(x^{(i)})$ for all $x^{(i)}$ in the ERM problem (2.1). Generally speaking, the regularization technique constrains the model class once and for all at training time in the hope of achieving evasion robustness at testing time. Distillation (distillation) is a generic technique for transferring the knowledge of one neural network classifier f into another classifier $\tilde{f}$ with a potentially different internal structure (Hinton et al. 2014). The authors of (Papernot et al. 2015) show the beneficial effect of distillation on flattening-out the prediction landscape of the distilled model $\tilde{f}$. However, this regularization effect does not translate into evasion robustness at testing time after careful evaluation (Carlini and Wagner 2016). The vaccination technique relies on the simple idea of training a classifier, generating evading instances for it, and re-training a new classifier on the original dataset augmented with the previously found but correctly labeled evading instances. A related technique is to randomly perturb the training set, focussing on features that are easy to change for an attacker. Iterated vaccination consists in repeating the vaccination technique for several rounds. When naively implemented, iterated vaccination increases the size of the training set at each iteration by a constant number of instances. In Chapter 5, we introduce a set of techniques that make iterated vaccination practical for tree ensemble classifiers and potentially very large datasets. Our technique, *adversarial boosting*, incrementally grows the ensemble classifier instead of the dataset.

As for any computer security problem, estimating the real-life effectiveness of an evasion defense is difficult. We use the question mark for papers that do not evaluate the evasion susceptibility of their proposed defense with a suitable attack strategy, or otherwise use a randomized cross-validation procedure to evaluate the accuracy of the proposed method. We use "low" to denote the case of papers which show that the proposed defense can still be evaded by a reasonable attack. In practical applications, it might be difficult to design a realistic evasion attack against a proposed defense. In those cases, the best evaluation strategy is to field the system and empirically observe its effectiveness against evasion attacks. As this is rarely a feasible approach, an interesting alternative is to use a temporally consistent validation procedure where the training data always predates the testing data, thereby simulating the passage of time and changing adversarial strategies. Papers that successfully test their defenses on temporally consistent benchmarks receive the "possibly" mark for this category. Finally, papers that successfully demonstrate robustness to evasion against the strongest possible adversary are rated "high".

Aside from thwarting evasions, a defense approach must not significantly decrease the performance of the detection pipeline. Papers which do not provide a regular accuracy evaluation are denoted with an interrogation mark. Papers that show a negative, neutral or positive impact on classification accuracy receive the corresponding rating.

As for attack strategies, we also estimate the computational cost in three increments: low, medium and high. Generally, feature level approaches have a low computational cost, as those approaches either build new features that are inexpensive to compute, or remove previously available features that give a disproportionate leverage to the adversary in crafting evading instances. Approaches that require training a new classifier are marked as medium complexity, except if model retraining is either very efficient (low rating) or very expensive (high rating).

Prior work	Domain	Type	Effective	Class. Impact	Cost
Dalvi et al. (2004)	generic, email spam	new model	?	?	?
Honglak and Ng (2005)	email spam	ψ-level	?	?	low
Bratko et al. (2006)	email spam	new model	?	positive	low
Bruckner et al. (2012)	generic, email spam	regularization	possibly	positive	high
Smutz and Stavrou (2012)	PDF malware	ψ-level, vaccination, perturbation	possibly	neutral	medium
Srndic and Laskov (2014)	PDF malware	ψ-level, vaccination, perturbation	low	negative	medium
Goodfellow et al. (2015)	generic, image recognition	regularization	low	positive	medium
Gu and Rigazio (2015)	generic, image recognition	regularization	low	neutral	medium
Papernot et al. (2015)	generic, image recognition	distillation	low	neutral	medium
This thesis, Chapter 3	comment spam	ψ-level	possibly	positive	low
This thesis, Chapter 4	generic	new model	low	positive	low
This thesis, Section 5.6	generic, image recognition	iterated vaccination	high	neutral	medium

Table 2.4: Contrasting prior work on defenses with the work presented in this thesis (last three rows).

Part I

Hardening Feature Extraction

Chapter 3

Comment Spam Case Study

3.1 Introduction

In the first part of this thesis, we examine a strategy for hardening a machine learning based detection pipeline at the feature extraction phase. That is, we design a more evasion-resilient feature extractor ψ while keeping a generic machine learning classifier f. As feature extraction is predominantly an application domain-specific task, we fully ground this chapter in a specific application. We choose to work on the problem of detecting comment spam on online social media for several reasons. First, comment spam is a high impact issue for online social media; second, comment spam presents highly adversarial characteristics, and third, comment spam has previously received only limited attention. The work in this chapter was first reported in (Kantchelian et al. 2012).

Comment Spam in Online Social Networks

Online social media have become indispensable, and a large part of their success is that they are platforms for hosting user-generated content. An important example of how users contribute value to a social media site is the inclusion of comment threads in online articles of various kinds (news, personal blogs, etc). Through comments, users have an opportunity to share their insights with one another. However, the medium presents a unique opportunity for abuse by criminals and other miscreants. Abusers use comment threads to post content containing links to spam and malware sites, as well as content that itself is considered as spam by users. If left unaddressed, abuse reduces the value of a social media site by reducing the proportion of legitimate comments, thereby leaving users less satisfied with their experience.

Many approaches have been proposed for detecting email spam, e.g. (Ramachandran et al. 2007; Sculley and Wachman 2007; Xie et al. 2008; J.-M. Xu et al. 2009). However, most of them are not directly applicable to detecting spam in social media. Spam in social media is different from email spam in several ways. The majority of spam email messages are generated by dumb botnets using certain predefined templates (Ramachandran and Feamster 2006). Large volumes of them contain similar content, format and target URLs (Zhuang et al. 2008), and display strong temporal regularity and correlation in their sending time (Ramachandran and Feamster 2006).

These properties make it relative easy to develop effective approaches to filter out email spam. Unlike email spam messages, social media spam messages, for example blog comment spam messages, are usually short and carefully crafted by humans, and even human experts have hard times to differentiate from legitimate ones. We also observe little temporal regularity in the posting of blog comment spam. These differences require the development of different detectors to filter out comment spam.

Social media spam also takes the advantage of the open nature of the blog comment space. Anonymous communication is allowed in most social media. A single spam message can potentially reach as many viewers as users of the social media. These spam messages usually target search engines to increase the pagerank of the advertised page as well as users.

Because of the volume and complexity of the data involved in detecting social media spam, approaches based on machine learning offer promising solutions since they scale beyond what a human expert can achieve. However, developing a machine learning based detection pipeline for comment spam face three fundamental challenges: first, we must develop a feature extractor ψ that both provide a strong signal for the classifier and is hard to temper with. Second, we must construct an algorithmic approach that can make effective use of the features. Third, we must develop an evaluation strategy. To be useful, the evaluation must both measure meaningful characteristics and be conducted in a setting as realistic as possible. This means using a sample set which accurately describes the real data and especially for our focus, shows evidence of evasion activities. Given a limited amount of hand-labeling resources and the challenge of fielding a given method and observing the reaction of real-world adversaries, this is usually a hard task.

Our approach

While it is possible to define blog comment spam as “any kind of undesirable content” analogous to personal email spam, such a single-user centric view is problematic in the context of an open medium where everybody is both entitled to contribute to the media and to access it. In particular, it is possible to find instances of comments where two legitimate users might disagree on whether the message is acceptable, as we show in the evaluation section. Moreover, the context of the surrounding website plays a crucial role in evaluating the undesirability of the comment (Mishne et al. 2005).

As the basis for our approach, we define spam as content that is uninformative in the information-theoretic sense. Intuitively, if the comments generated by a particular user are highly redundant, then there is a high likelihood that these messages will appear as undesirable to the other users of the social media. By carefully calibrating the compression ratio output of an off-the-shelf data compressor, the *content complexity* metric we develop quantifies this redundancy in an uniform manner for variable length texts. Once constructed, we can almost immediately obtain meaningful features for blog comment spam detection using this metric. Because content complexity operates at the character-level over an aggregation of messages, it is relatively insensitive to classical evasion strategies that plague regular content-based filters. In particular, when aggregated in a sensible manner, neither adversarial misspellings of words nor templated spam messages exhibit enough diversity to significantly perturb the content complexity metric. We discuss the limitations of our information theoretic approach in Section 3.6.

In addition to the content complexity features, we introduce a latent variable model that can tolerate noisy labels. For practitioners who want to use machine learning methods, acquiring enough labeled training data is one of the biggest challenges. Sometimes they may gain access to feeds or services that provide machine-labeled data that can be noisy (i.e., with the ground-truth label for the data being different from the label provided). In an adversarial context, this noise-tolerance is important because it may make our approach robust to mislabeled outliers deliberately introduced. As such, we adapt our model training to be tolerant of noisy labels. We use a latent variable model to handle noisy labels and enhance the detection accuracy of the content complexity features.

Our approach provides contributions on three fronts:

1. We introduce *content complexity* features for characterizing the IP addresses, usernames, and embedded URL hostnames associated with each comment;
2. We adopt a latent variable model for training a classifier with complexity features (with non-linear expansion) that can *tolerate noisy labels* in our data sets;
3. We conduct a rigorous, temporally-consistent evaluation of our method, leading to semi-normalized precision-recall curves which we believe are more telling than both receiver operating characteristic curves or their associated single-dimension area under the curve metric.

Related Work

User-generated content (e.g., comments and reviews) is widely available on blog and online shopping websites, and online social networks. The authors of (Mishne and Glance 2006) analyze various aspects of Web blog comments in their 2006 study, which among other findings, showed that blog comments could be as crucial as the original blog post in providing original, searchable content to the article. Brennan et al. use metadata, for example user activity, reputation, comment time, and posts' contents to predict the community ratings of the Slashdot comments (Brennan et al. 2010).

Ranking and rating help to promote high-quality comments and demote low-quality ones. Using a large corpus of Digg stories and comments, the authors of (Hsu et al. 2009) collect a set of features related to user, content, and popularity to train a Support vector Machine to rank comments according to quality. The authors find that the tandem of user and content based features are among the most effective ones. Their measure of content complexity is different from ours because they compute the entropy of a single comment message, whereas the study in this paper computes messages over a set of comments grouped. The authors of (B.-C. Chen et al. 2011) show that the quality of a comment is almost uncorrelated to the ratings of comment, and propose a latent factor model to predict comment quality based on its content, author reputation, agreement in opinions, etc. In (Mishra and Rastogi 2012), the authors apply semi-supervised learning techniques to address the user bias issue in comment ratings using information from user-comment graph.

Using data compression for spam filtering is not new. The authors of (Bratko et al. 2006) propose adaptive data compression models for email spam filtering. They train the classifier by building two compression models from the training corpus, one from spam examples and one from legitimate examples, and then predict the label of new instances using a minimum description length principle (Barron et al. 1998).

In (Mishne et al. 2005), the authors study the feasibility of using unigram language models for detecting off-topic link spam blog comments. They use Kullback-Leibler divergence between language model of the blog post and that of the comment to predict whether a comment is link spam. In (Shin et al. 2011), the authors study comment spam on a research blog using various content specific and host-based features. Their approach showed significant performance but was limited to only one particular blog whereas our data are more diverse including a wide range of personal and commercial sites. In (Gao et al. 2010), the authors quantify and characterize spam campaigns on online social networks using a dataset of "wall" messages between Facebook users. Their approach first groups together wall posts that share either the same URL or strong textual similarity using a graph-based clustering technique, and then apply threshold filters based on the wall post sending rate of user accounts and time correlation within each subgraph to distinguish potentially malicious clusters from benign ones.

The authors of (Lee et al. 2010) propose a honeypot-based approach for uncovering social spammers in online social network. They deploy social honeypots for harvesting deceptive spam profiles from social networking communities, and create spam classifier using machine learning methods based on a variety features derived from user demographics (e.g., age, gender, location), user contributed content (e.g., blog posts, comments, tweets), user activity features (e.g., posting rate, tweet frequency), and user connections (e.g., number of friends, followers, following).

Email spam problem has been extensively studied in recent years. One category of approaches are based on machine learning techniques. In (Sculley and Wachman 2007) the authors show that linear classifiers produce state-of-the-art performance for online content-based detection of spam on the web (e.g., spammy email, comments, blogs). To relieve the burden of labeling large-scale data, the use of active semi-supervised learning method for the training spam classifier has also been proposed (J.-M. Xu et al. 2009). The author's approach can leverage both unlabeled emails and asynchronous human feedback to build high performance classifiers with small amount of labeled data. Instead of using content-based information, one can also develop a spam filtering system to classify email senders based on their sending behavior (Ramachandran et al. 2007). The authors of (Xie et al. 2008) characterize spamming botnets using information from both spam payload (e.g., embedded URLs) and spam server traffic, and develop a signature-based framework to detect botnet-based spam emails and botnet membership.

The rest of this chapter is organized as follows. Section 3.2 introduces the content complexity metric and explains the construction of our feature vectors for comment spam detection. Section 3.3 describes the latent variable model we use for classification and tolerating noisy labels. Section 3.4 describes the data set and the methods we use for the evaluations in Section 3.5. We finish this chapter with a list of open problems in Section 3.6.

3.2 Content Complexity

We now turn to the construction of the content complexity metric upon which we build features for comment spam detection.

Intuitively, content complexity describes the amount of redundancy in the content associated with a string. While remaining language-agnostic, it is normalized for the underlying compression method, natural language and string length. In everything that follows and without loss of generality, $s \in \{0; 1\}^*$ is a binary string and $|s|$ designates its length, C is a lossless data compression algorithm.

The basis for the content complexity metric is the compression ratio $|C(s)|/|s|$. However, we expect the compression ratio of a string to be sensitive to the original string length (e.g., we expect the complexity of a long essay to be lower than the complexity of a short paragraph). Because we want to be able to compare complexity values between strings of different lengths, we introduce a normalization factor $h(n)$, a parametrized function which models the expected compression ratio of strings of length n. The addition of $h(n)$ allows the content complexity of a string to be calculated independently of its length.

Modeling the Compression Ratio

Let $r_C(s) = |C(s)|/|s|$ be the compression ratio (or the complexity rate) of string x under compressor C. Let $s_{1:n}$ denote the first n bytes subsequence of s. For any well behaved s and compression algorithm C, we use the following function $h(n)$ to approximate $r_C(s_{1:n})$:

$$\begin{gathered} h(n) = \alpha + A \log n / n^\gamma + B/n, \\ \alpha, A, B > 0, \quad 0 < \gamma < 1, \end{gathered} \tag{3.1}$$

where α, A, B, γ are constants which depend on the probability source emitting s alone.

The term B/n represents the fixed-size overhead of off-the-shelf compression algorithms, and is an extension to Schurmann's model to help model the expected compression ratio for small-to-medium values of n (Schurmann and Grassberger 1996). For example, compressing a zero-length string with Lempel-Ziv-Markov chain algorithm (LZMA) (Pavlov 2007) produces a 15 byte output. The term is asymptotically negligible for large values of n.

The first two terms $\alpha + A \log n / n^\gamma$ describe a power-law convergence to a fixed value α which can be interpreted as an upper bound on the entropy rate of s.

Finally, while it is possible to give a precise mathematical statement for what we mean by a "well behaved" sequence by requiring stationarity and ergodicity of the stochastic process, we consider in this work general sequences from natural language for which such strong assumptions seem unwarranted. Furthermore, the model is essentially a postulate which is only backed up *a posteriori* by the following experimental evidence.

Natural Language Complexity Rates

For the rest of this work, we fix the compression function to be LZMA. One reason for this choice is that LZMA can be set up to use a very large compression block size - or equivalently a very large dictionary size - which makes it able to capture long-range dependencies in its input. For each of the six languages presented in Figure 3.1, we randomly select a dozen plain text UTF-8 e-books from Project Gutenberg. We minimally pre-process each file by discarding its header and footer data which contain various copyright information in English, irrespective of the actual document language. Then, for each text, we compute the sequence of compression ratios on initial subsequences of increasing size and plot the resulting graph.

We finally superimpose the predicted compression ratio by the above model, where the four parameters have been obtained by a standard Levenberg-Marquardt (Levenberg 1944; Marquardt 1963) optimization on the non-linear least squares formulation. The initial parameters values are $(\alpha, A, B, \gamma) = (0, 1, 1, 0.5)$, but the final result is independent of these values on a large domain. We fit the model using the data of all six languages so that we obtain a single set of parameters.

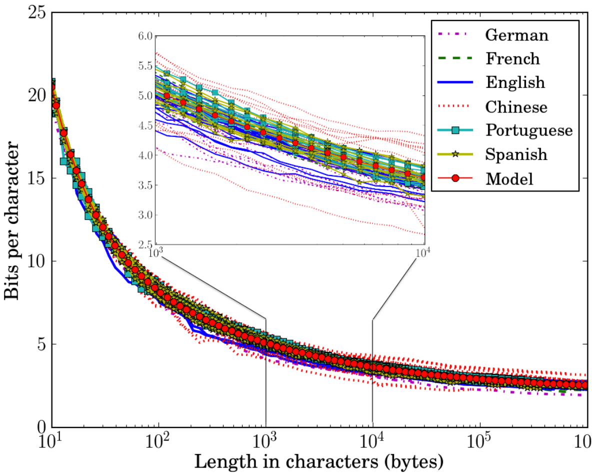


Figure 3.1: LZMA compression ratios $|C(s)|/|s|$ *for prefixes of ebooks in six languages, with a single fitted model (*$R^2 = 0.995$*). Each line represents one ebook. Notice the lower compressibility of Chinese.*

The optimal parameters derived from our dataset are:

$$(\alpha, A, B, \gamma) = (2.23, 7.13, 120, 0.419) \tag{3.2}$$

Notice that the extrapolated LZMA entropy rate on infinite length sequences is $\alpha = 2.23$ bits per byte. This is compatible with although much higher than Shannon's experiments (Shannon 1951) suggesting an average entropy rate of 1.3 bits per character for English texts. Remarkably, LZMA's stable behavior is well captured by our simple model. While European languages are particularly well described, the Asian language samples of our dataset exhibit a much higher variance.

We can finally define the content complexity metric of a string s as $Q(s) = |C(s)|/|s| - h(|s|)$, where h is defined in Eqn (3.1), and its parameters are previously computed by Eqn (3.2). The content complexity $Q(s)$ is a real number. Intuitively, it represents the intrinsic "informativeness" of the string s independent of its length. A low value of $Q(s)$ means that s contains very little new information and has a lot of redundancy. A high value of $Q(s)$ means that x contains a lot of new information and has little redundancy. Note that our definition of “informative” does not exactly correspond to the colloquial use of the term. Nonetheless, as we discuss below, our definition is highly effective at identifying commercial and other spam.

From Complexity to Detection Features

One important question to answer is whether the natural language as encountered in blog comments is as well described by the same model h as for e-books. Figure 3.2 gives evidence of a positive answer. Each point on this plot represents a username for which all the contributions are concatenated together in a single string and the compression ratio is subsequently computed. The predicted natural language complexity h is represented in solid black. The model curve still describes fairly accurately the bulk of the distribution, except for the fact that the entropy rate of blog comments seems slightly lower than the one of e-books. More interestingly, we observe a relatively clear separation between ham and spam on this graph.

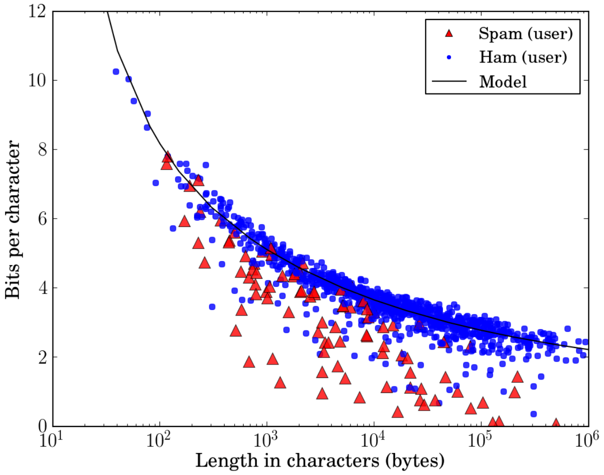


Figure 3.2: A random sampling of a thousand usernames with two associated comments or more. For each user, all her comments are concatenated and the compression ratio is computed. The predicted compression ratio h learned on the e-books is indicated in solid black. The labels are propagated from the comments to the usernames by a simple zero-tolerance rule: a user is labeled as spam if and only if one of her comments is labeled as spam. Notice the occurring separation between ham and spam.

The steps for generating our features are as follow. First, we minimally normalize the messages by removing periodic runs of characters, for a period of at most four chars. By convention, we only keep the first two runs. For example, `ahahahah` becomes `ahah`, `ooooh` becomes `ooh` but `ahAHaHaHAHAh` remains unchanged. Such periodic runs artificially lower the content complexity metric and are a non negligible source of false positives when left unaddressed.

Second, we choose an aggregation field between messages. In this work, we form the following four groups:

1. messages which share the same username,
2. messages which share the same host name in any linked URL,
3. messages which are posted to the same permalink,
4. messages which come from the same IP address, within a given time period.

The reason why we aggregate on multiple dimensions and not just on a per username basis is that a consequent portion of the comments of our dataset are anonymous, and that an adversary can easily generate a new user account for every new comment, leading to singleton groupings. This evasion strategy is made more difficult by aggregating and scoring across IP and including URL host names.

For the URL extraction, we use a relatively naive regular expression which matches top-level domains. To avoid IP aliasing, the aggregation by IP is parametrized by a timing parameter Δt, such that if the time delta between two posts coming from the same IP address is smaller than Δt, the messages are grouped together. The full IP clustering is obtained by transitively applying the property: two messages a, b are in the same IP grouping if and only if there exists a sequence of messages staring at a and finishing at b sharing the same IP and such that the consecutive time deltas are smaller than Δt. Δt is taken to be three hours.

A further cleaning-up step we take is removing duplicate messages within groups. For our dataset, we do this by relying on the presence of an `update` flag. We only keep the latest version of an updated comment.

Finally, we compute the content complexity metric for all the groups of two messages or more, and we back-propagate the results to the messages themselves. In the case of the host name grouping, when a message contains several linked URLs and thus belongs to several host name groupings, we back-propagate the lowest content complexity metric among all these groups. If a grouping resulted in a singleton, we assign the normal content complexity zero to the corresponding feature.

We call this set of features F_C, for *complexity* features. Thus, there are exactly four F_C features per message. We also define F_{LGS} for *log-group-size* which as their name suggests give the logarithm of the grouping size (four features per message again), and F_{dG} for *is-defined-group* which are four binary features indicating whether the associated grouping resulted in a singleton or not.

3.3 Latent Variable Model

We now turn to the design of the classifier. As our base component, we use a linear model with the machinery of logistic regression (LR), which accommodates well to unknown dependencies between features. Besides, a linear model is a good classifier candidate in this case: classification is

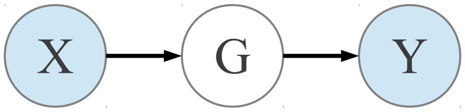


Figure 3.3: Graphical model for our learning approach

intuitively monotonic in the complexity features: a low content complexity score is highly indicative of bulk and thus spam messages.

More formally, our approach of classifying social media comments trains a model using a dataset of n points: $\{(x^{(1)}, y_1), (x^{(2)}, y_2), \ldots, (x^{(n)}, y_n)\}$. For comment i in the data set, $x^{(i)}$ is the vector representing its associated features and $y_i \in \{0, 1\}$ represents the label (our convention is zero for ham, one for spam).

Furthermore, the label y_i might be a noisy representation of the example's *ground truth* label $g_i \in \{0, 1\}$. For example, a spam comment could be mislabeled as benign in the training set (label $y_i = 0$) when its ground truth label is $g_i = 1$. The goal of classification is to train a model so that given a feature vector $x^{(i)}$ we can predict g_i with high accuracy. Because the ground truth label is not observed directly (only x and the noisy y are observed), we model g as a latent variable and derive a latent version of LR similar to the approach proposed in (Raykar et al. 2010).

Model Description

Here we describe the probabilistic framework for the latent model, which is illustrated in Figure 3.3. The variable X represents the distribution of messages in the feature space, G represents ground truth labels, and Y represents the noisy labels. We assume both X and Y are visible and G is hidden. The conditional probabilities which define the model are $P(G|X)$ (for inferring the ground truth label given a data point) and $P(Y|G)$ (for modeling the noise in the dataset's labels).

We parametrize the conditional probability $P(G|X)$ using the usual logistic regression model as follows:

$$P(G = g|X = x) = \sigma(x^{\intercal} w)^{g} \sigma(-x^{\intercal} w)^{(1-g)} \tag{3.3}$$

where σ is the logistic function: $\sigma(q) = (1 + e^{-q})^{-1}$. We mention but notationally omit that we append a dummy constant unitary dimension to all feature vectors x to handle bias.

We define the noise model $P(Y|G)$ as a mixture of Bernoulli distributions:

$$P(Y = y|G = g) = \alpha^{gy}(1 - \alpha)^{g(1-y)}(1 - \beta)^{(1-g)y}\beta^{(1-g)(1-y)}$$

Because y and g are either zero or one, the exponents act as indicator functions that select a probability based on whether the ground truth label and data label match. For example, α is the probability that a spam blog comment is labeled correctly ($g = 1$ and $y = 1$), $(1-\alpha)$ is the probability that a spam blog comment is labeled incorrectly ($g = 1$ and $y = 0$), $(1 - \beta)$ is the probability that a ham blog comment is labeled incorrectly ($g = 0$ and $y = 1$), and β is the probability that a ham blog comment is labeled correctly ($g = 0$ and $y = 0$).

Learning

The parameters we have to learn for our model are the classification weight vector w, and the noise parameters α, β. We use an expectation-maximization (EM) procedure (Dempster et al. 1977) to maximize the model's log-likelihood $\mathcal{L}(w, \alpha, \beta) = \sum_i \mathcal{L}_i(w, \alpha, \beta)$, where the log-likelihood of each data point $(x^{(i)}, y_i)$ is

$$\mathcal{L}_i(w, \alpha, \beta) \triangleq \begin{array}{l} g_i y_i \log \alpha + g_i(1 - y_i) \log(1 - \alpha) + \\ (1 - g_i) y_i \log(1 - \beta) + (1 - g_i)(1 - y_i) \log \beta + \\ g_i \log \sigma(x^{(i)\mathsf{T}} w) + (1 - g_i) \log \sigma(x^{(i)\mathsf{T}} w). \end{array} \tag{3.4}$$

In the expectation step, we want to compute the expected value for each hidden ground truth label g_i in our dataset using existing estimates for conditional probabilities in our model. Because g_i is a binary value, we have $\mathbb{E}[g_i | x^{(i)}, y_i] = P(g_i = 1 | x^{(i)}, y_i)$ which we calculate as follows:

$$P(g_i = 1 | x^{(i)}, y_i) = \frac{P(g_i = 1 | x^{(i)}) P(y_i | g_i = 1)}{\sum_{g \in \{0,1\}} P(g | x^{(i)}) P(y_i | g)}. \tag{3.5}$$

Then, we assign $\hat{g}_i \leftarrow \mathbb{E}[g_i | x^{(i)}, y_i]$, substitute $\hat{g}_i$ for g_i in Equation (3.4), and then proceed to the maximization step.

In the maximization step, we reassign the model parameters w, α, and β to maximize the log-likelihood. Using the logistic regression model and the L-BFGS optimization routine (Richard H. Byrd et al. 1995), we reassign the parameter vector w. The noise parameters α and β are reassigned as follows:

$$\alpha \leftarrow \frac{\sum_i \hat{g}_i y_i}{\sum_i \hat{g}_i} \tag{3.6}$$

$$\beta \leftarrow \frac{\sum_i (1 - \hat{g}_i)(1 - y_i)}{\sum_i (1 - \hat{g}_i)} \tag{3.7}$$

A good initialization is a key to a successful run of the EM algorithm. Experimentally, we found that initializing w to be the result of the plain logistic regression on the dataset and setting $\alpha = \beta = 0.5$ provides the best results.

Finally, we stop the EM loop as soon as we exceed 300 iterations or the relative L_1 difference of two consecutive w parameter vectors is within 1%, i.e.,

$$\frac{|w^{(i)} - w^{(i-1)}|}{|w^{(i-1)}|} \leq 0.01$$

Feature Expansion

To capture potential non-linear effects among features, we use polynomial basis functions over those features. Specifically, if $x = (x_1, x_2, \ldots, x_d)$ is our initial feature vector of d features, we expand it to all the terms in the expansion of the quadratic polynomial $(1 + x_1 + \ldots + x_d)^2$. We define $\text{expan}(x, 2)$ as the mapping from the original data vector x to the quadratic polynomial expansion.

Formally:

$$\text{expan}(x, 2) = (1) \cup (x_i)_i \cup (x_i x_{i'})_{i \leq i'}.$$

Expanding a feature vector of size d results in a new vector of size $\Theta(d^2)$, thus we can only use this strategy when the initial number of features is small.

Kernel logistic regression can be also used to handle non-linear effects among features. However, it usually results in more complex models and requires more computation budget to train a model and make predictions than the linear method. We leave it as future work to explore the feasibility of applying kernel logistic regression to our problem.

3.4 Evaluation Method

In this section, we explain our evaluation method. We start by describing our dataset and motivate a sampling strategy for efficient hand-labeling. We conclude by explaining our actual labeling process.

Dataset

The dataset we use for our evaluations comes from a provider that aggregates comment threads from a variety of social media platforms, such as personal, political or business oriented blogs, news websites, various entertainment websites, etc. The collection time period is four months between December 2011 and March 2012. Non-English languages are present at a significant level: we often encounter comments in Spanish, Portuguese, Chinese, Japanese or French. The comment data also comes with machine-generated labels for each comment which contain some error. The provider is a well reputed source which implement cutting-edge statistical filtering techniques.

In practice, the dataset resides in a Hadoop cluster where each record has the following fields:

```
timestamp, user_id, end_user_ip, article_permalink, content, spam_label
```

For a rigorous evaluation of our algorithms, we divide this dataset into two consecutive equal length time periods of two months each. The first time period will always serve as the training set (set A), while the second will be used for scoring only (set B). In particular, when computing the groupings for the content complexity metrics, we never aggregate together two messages coming from distinct subsets: future information is not available when training, and past information is not available when scoring.

This temporally-consistent style of evaluation also ensures that existing evasion attacks in the dataset are not denatured. Indeed, for the alternative random cross-validation scheme, training and testing instances are randomly drawn without replacement from the dataset, negating the effect of time and flattening out the surprising, adaptive or otherwise hard to predict observations.

A summary of the split dataset is presented in Table 3.1. For equal time periods, the scoring dataset is significantly larger than the training dataset. We explain this fact by the rapid gain in popularity and expansion of the associated web service. Note that the anti-aliasing process for grouping by IP can produce several distinct groups sharing the same IP, but at different times.

Characteristic	Training set *A*	Scoring set *B*
Time period	12/01/2011 01/31/2012	02/01/2012 03/31/2012
Number of comments	25m (420k/day)	40m (660k/day)
Number of distinct user ids	990k	1.5m
Distinct anti-aliased IP addresses	3.9m	6.5m
Number of distinct article permalinks	1.4m	2.3m
Distinct hosts from linked URLs	99k	150k
Labeled spam	1.8%	3.6%

Table 3.1: Learning and training set characteristics.

An Unbiased Sampling Strategy

While we do have the provider's labels for each of sets *A* and *B*, we can only use them for training. Indeed, we know and we observe in the following evaluations that these labels display a non negligible false positive rate. Since our goal is to compare our methods both against themselves and the provider, we must establish a form of ground truth by manually labeling carefully chosen samples from *B*.

Also, notice that we still want to use the provider's labeling for training, as a way to minimize the hand-labeling labor and test the latent logistic regression model in the wild.

While remaining unbiased, our sampling strategy is driven by two stringent constraints:

- the high cost of obtaining ground truth labels and
- the scarcity of positive (spam) instances in the data.

Because labeling blog comments requires looking up contextual information, e.g., the actual blog post, surrounding comments, user profile and user past activity, translating the contents in English when necessary, it is a rather time consuming task. Under such circumstances, a naive uniform sampling over *B* is inefficient.

The problem is exacerbated because only a small fraction of the comments are actually spam. In practice, this means that naive uniform sampling tends to have a strong bias towards extracting ham comments.

The approach we choose relies on the observation that if one is just interested in the precision score, i.e., the proportion of spam among all the flagged instances, then restricting the sampling to be uniform only over the flagged instances is sufficient. This is expressed by the following basic result.

Let $l : B \to \{0; 1\}$ the ground truth labeling (0=ham, 1=spam). Let τ be a detection threshold and f a classifier. For notational convenience, we define $f_\tau(x) = \mathbb{I}_{f(x)>\tau}$ to be our decision function. τ-level precision is defined as:

$$p_\tau = \frac{\sum_x f_\tau(x) l(x)}{\sum_x f_\tau(x)}$$

Let $B^+ = \{x \in B, l(x) = 1\}$ be the subset of spam messages and $\chi_\tau^+ = \{x \in B, f_\tau(x) = 1\}$ be the flagged users at detection level τ.

Proposition 3.4.1. Let $\tilde{\chi}_\tau^+ \subset \chi_\tau^+$ a uniformly selected subset of size n. The following is an unbiased estimator of p_τ:

$$\tilde{p}_\tau = \frac{\sum_{x \in \tilde{\chi}_\tau^+} l(x)}{n} \quad \text{and} \quad \operatorname{Var} \tilde{p}_\tau = \frac{|\chi_\tau^+| - n}{n(|\chi_\tau^+| - 1)}(p_\tau - p_\tau^2) \leq \frac{1}{4n}.$$

Proof. We first show that the estimator $\tilde{p}_r$ is unbiased:

$$\begin{aligned}
\mathbb{E}[\tilde{p}_\tau] &= \binom{|\chi_\tau^+|}{n}^{-1} \sum_{|\tilde{\chi}_\tau^+|=n} \frac{\sum_{x, x \in \tilde{\chi}_\tau^+} l(x)}{n} \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n} \sum_{|\tilde{\chi}_\tau^+|=n} \sum_{x \in B} l(x) 1_{x \in \tilde{\chi}_\tau^+} \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n} \sum_{x \in B} l(x) \sum_{|\tilde{\chi}_\tau^+|=n} 1_{x \in \tilde{\chi}_\tau^+} \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n} \sum_{x \in B} l(x) \binom{|\chi_\tau^+| - 1}{n - 1} 1_{x \in \chi_\tau^+} \\
&= \frac{1}{n} \sum_{x \in B} l(x) 1_{x \in \chi_\tau^+} \frac{n}{|\chi_\tau^+|} \\
&= \frac{1}{|\chi_\tau^+|} \sum_{x \in B} l(x) 1_{x \in \chi_\tau^+} \\
&= p_\tau
\end{aligned}$$

The variance of our estimator is:

$$
\begin{aligned}
\mathbb{E}[\tilde{p}_\tau^2] &= \binom{|\chi_\tau^+|}{n}^{-1} \sum_{|\tilde{\chi}_\tau^+|=n} \left(\frac{\sum_{x, x\in\tilde{\chi}_\tau^+} l(x)}{n} \right)^2 \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n^2} \sum_{|\tilde{\chi}_\tau^+|=n} \left(\sum_{x\in B} l(x) 1_{x\in\tilde{\chi}_\tau^+} \right)^2 \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n^2} \sum_{|\tilde{\chi}_\tau^+|=n} \sum_{x,y\in B} l(x)l(y) 1_{x\in\tilde{\chi}_\tau^+} 1_{y\in\tilde{\chi}_\tau^+} \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n^2} \sum_{x,y\in B} l(x)l(y) \sum_{|\tilde{\chi}_\tau^+|=n} 1_{x\in\tilde{\chi}_\tau^+} 1_{y\in\tilde{\chi}_\tau^+} \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n^2} \sum_{x,y\in B} l(x)l(y) \left(\binom{|\chi_\tau^+|-1}{n-1} 1_{x,y\in\chi_\tau^+} 1_{x=y} \quad + \binom{|\chi_\tau^+|-2}{n-2} 1_{x,y\in\chi_\tau^+} 1_{x\neq y} \right) \\
&= \frac{1}{n|\chi_\tau^+|} \sum_{x,y\in B} l(x)l(y) 1_{x,y\in\chi_\tau^+} \left(1_{x=y} + \frac{n-1}{|\chi_\tau^+|-1} 1_{x\neq y} \right) \\
&= \frac{1}{n|\chi_\tau^+|} \Bigg(\frac{n-1}{|\chi_\tau^+|-1} \sum_{x,y\in B} l(x)l(y) 1_{x,y\in\chi_\tau^+} \\
&\quad + \frac{|\chi_\tau^+|-n}{|\chi_\tau^+|-1} \sum_{x\in B} l(x) 1_{x\in\chi_\tau^+} \Bigg) \\
&= \frac{1}{n(|\chi_\tau^+|-1)} \left[|\chi_\tau^+|(n-1)p_\tau^2 + (|\chi_\tau^+|-n)p_\tau \right]
\end{aligned}
$$

Hence:

$$
\begin{aligned}
\operatorname{Var} \tilde{p}_\tau &= \mathbb{E}[\tilde{p}_\tau^2] - \mathbb{E}[\tilde{p}_\tau]^2 = \mathbb{E}[\tilde{p}_\tau^2] - p_\tau^2 \\
&= \frac{|\chi_\tau^+|-n}{n(|\chi_\tau^+|-1)} (p_\tau - p_\tau^2)
\end{aligned}
$$

□

If we had a single classifier f to evaluate, we could start by finding a detection threshold τ_0 such that $\frac{|\chi_{\tau_0}^+|}{|B|} = 5.66\%$ for instance, meaning that the classifier flags 5.66% of the dataset at level τ_0. This is justified by the fact that the provider's labeling which serves as a baseline comparison is such that $\frac{|\chi^+|}{|B|} = 3.62\%$, so that we need at least the same proportion of flagged instances for comparison. Adding a reasonable safety margin gets us to 5.66%.

From there, we uniformly sample n instances $\tilde{\chi}_{\tau_0}^+ \subset \chi_{\tau_0}^+$ to obtain the base evaluation sample for f. Notice that for the same classifier f and $\tau > \tau_0$, any subset $\{x \in \bar{\chi}_{\tau_0}^+, f(x) > \tau\}$ is also a uniform sample in χ_τ^+, albeit of a smaller size.

Besides the fact that such sampling does not directly yield the recall or equivalently the false negative rate of the classifier, a major disadvantage is that the sampling is completely dependent on the classifier. A sampling that is uniform for one classifier has no reason to be uniform for another one. Thus, one practical issue is that it is difficult to sample and start labeling before the classifier is defined.

Concerning the recall issue, we notice that it is sufficient to multiply the estimated precision $\tilde{p}_\tau$ by the volume of flagged instances $|\chi_\tau^+|$ to obtain an unnormalized measure which is directly proportional to the recall of the algorithm. As $|\chi_\tau^+|$ is exactly known, the uncertainty on the unnormalized recall is simply $\sigma(\tilde{p}_\tau)|\chi_\tau^+|$. To obtain a dimensionless number for the evaluations, we use $\tilde{p}_\tau \frac{|\chi_\tau^+|}{|B|}$ as our unnormalized recall measure.

Finally, we build our evaluation dataset using the following strategy. Fix a uniform sampling rate $0 < r < 1$ and a flagged volume $0 < v < 1$. Let $f^1, \ldots, f^K$ be K classifiers we are interested in evaluating. For each classifier k, compute by binary search the minimal detection threshold τ_0^k such that $\{x \in B | f^k(x) > \tau_0^k\}/|B| \approx v$. Uniformly sample with rate r in subset:

$$\{x \in B | f^1(x) > \tau_0^1\} \cup \cdots \cup \{x \in B | f^K(x) > \tau_0^K\}.$$

The resulting sample provides by construction an unbiased evaluation sample for each classifier w.r.t. the $\tilde{p}$ measure, provided no classifier k is operated at detection thresholds lower than τ_0^K. In practice, we choose $r = 0.06\%$ and $v = 5.66\%$, which corresponds to a minimum of $rv|B| = 1358$ samples to label.

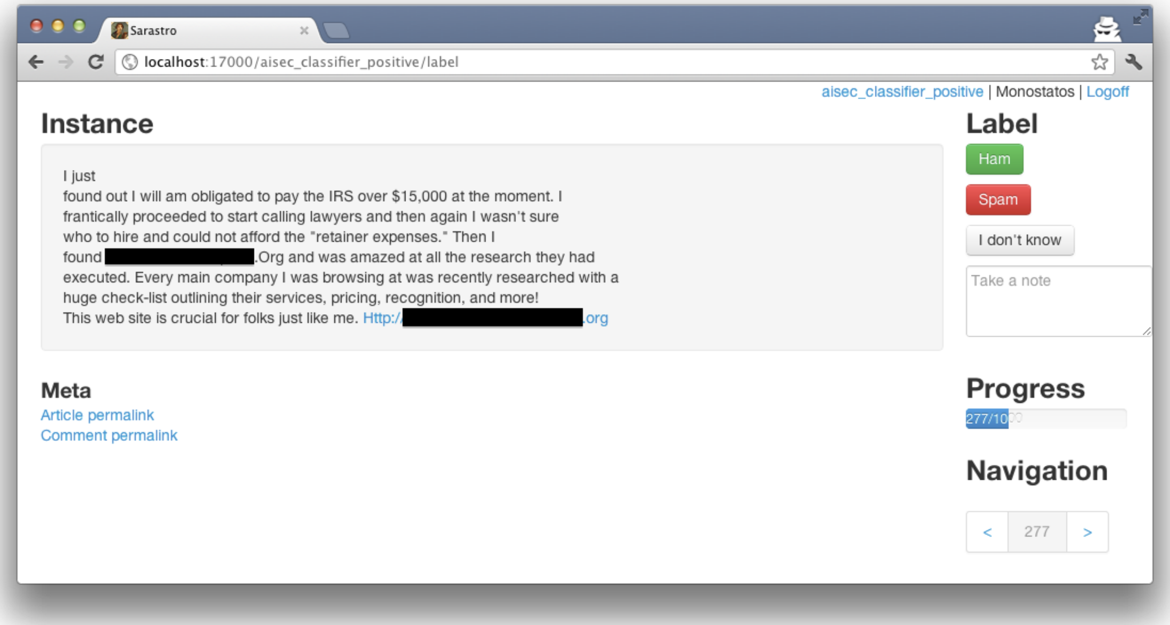
Labeling Process

Once constructed, we must hand-label the sampled set. To this end, we wrote a simple Python WSGI web application managing simultaneous labelers and datasets. Figure 3.4 shows the labeling screen the labeler is presented after logging in and selecting a task. In the labeling screen, the possible actions for the user are: (1) assign one of the three labels $\{$*ham*, *spam*, *I don't know*$\}$ along with an optional short note on the instance; (2) browse back or forward to correct a label; (3) look at the comment in its context.

The *spam* label is assigned to a comment when at least one of the following is true.

1. Comment links or refers to a commercial service (most of the time luxury, beauty, pharmaceutical, dating, or financial offers) and appears to be completely unrelated to the comments thread. No URL needs to be present as the spammer can include it in the account profile, or simply ask the users to search for a particular term.
2. Comment is a generic “thank you” or “nice blog”, with the intent of boosting the user account reputation (when available, we examine user's history).

Each comment gets three potentially conflicting labels from the three labelers, who used much more or even completely different information for labeling than what the algorithm uses. With three labelers and three labels, an instance label can only be one of the following. It is unanimous when



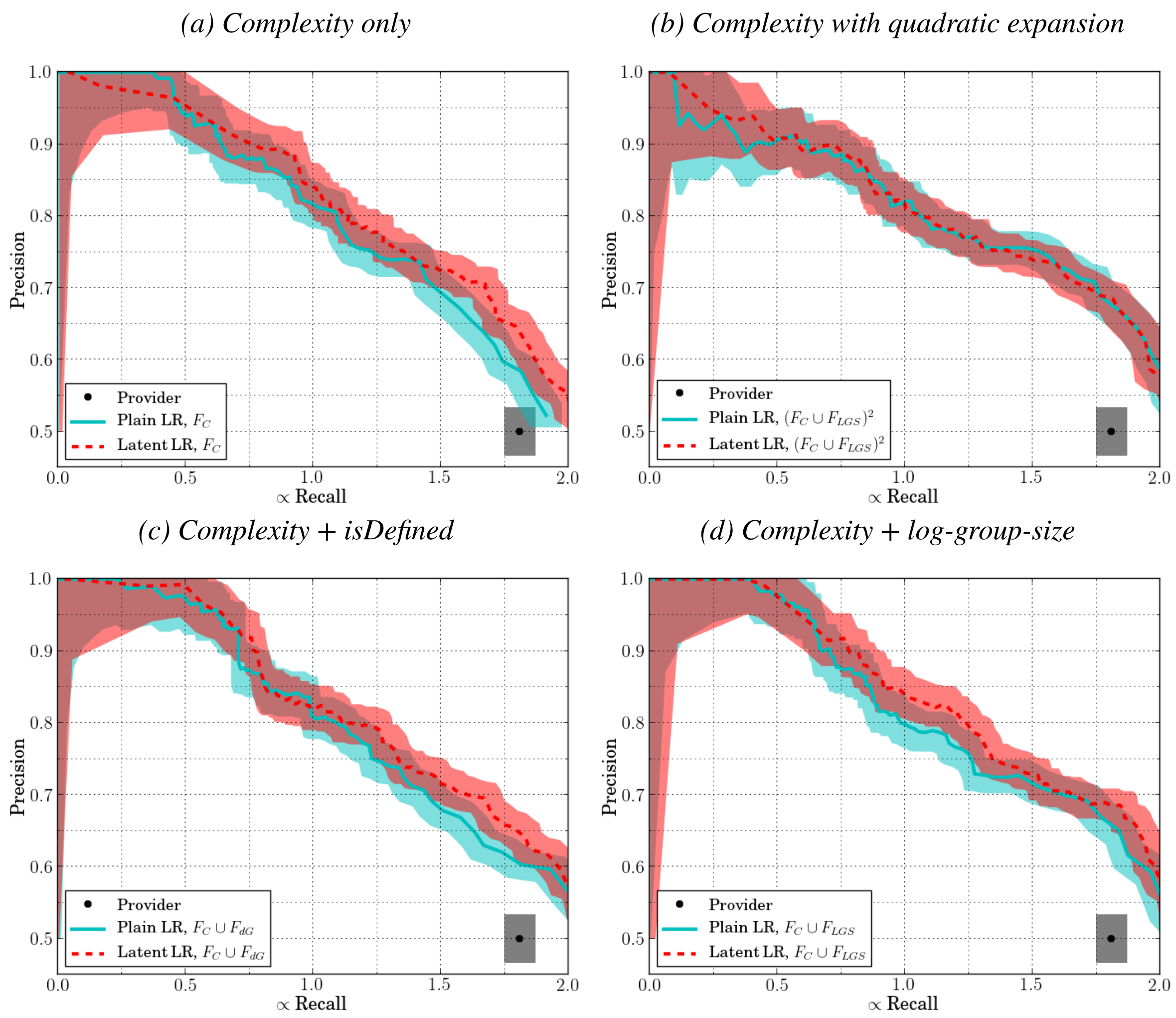
I just
found out I will am obligated to pay the IRS over $15,000 at the moment. I
frantically proceeded to start calling lawyers and then again I wasn't sure
who to hire and could not afford the "retainer expenses." Then I
found [redacted].Org and was amazed at all the research they had
executed. Every main company I was browsing at was recently researched with a
huge check-list outlining their services, pricing, recognition, and more!
This web site is crucial for folks just like me. Http:/[redacted].org

Figure 3.4: The developed user interface for testing set labeling. Labeler Monostatos has already labeled 277 over 1000 instances in task aisec_classifier_positive.

the labelers choose the same label, a majority when exactly two labelers choose the same label, and conflicting when all three labelers choose a different label. Table 3.2 summarizes the labelers disagreement on all the evaluated instances. We briefly tried to resolve conflicts, but quickly backtracked as the process is very time consuming. Instead, we used the majority label when one was available, and we treated conflicts and *I don't know* labels as *spam*. We also experimented with turning the uncertain labels to *ham* or simply discarding them from the dataset at the risk of breaking the uniformity of the sample. None of the policies noticeably changed the results and the conclusions.

Label status	number of instances	proportion
Any	2349	100%
Unanimity	1575	67%
Majority	728	31%
Conflicted	46	2%
I don't know	28	1%

Table 3.2: The final evaluation sample characteristics. Most of the comments have an unanimous label while a tiny fraction result in intra-labelers conflicts.


(a) Complexity only

(b) Complexity with quadratic expansion

(c) Complexity + isDefined

(d) Complexity + log-group-size

Figure 3.5: Precision-recall plots for Plain LR and Latent LR using different combinations of features, with standard deviations for both axis.

3.5 Evaluation

We evaluate the approach of using our latent variable model with content complexity features associated with social media comments from our data set. Over the course of the evaluation, we want to answer the following questions. How effective are content complexity features for classification? Does the noise-tolerant latent variable model provide an improvement over standard logistic regression which is not noise-tolerant? And which combinations of complexity features provide the most accurate classification?

Figure 3.5 shows the precision-recall curves for the classification algorithms, comparing Plain LR to Latent LR using four different combinations of features. In all figures, the precision and recall of the data provider's labels is shown for reference as a dot because the labels are discreet "ham/spam" labels. The scaling of the x- and y-axis of all figures is the same. The filled area around the curve is an upper bound of a standard deviation above and below the expected precision and recall for a given threshold, as given by proposition 3.4.1. Notice the large precision variance in the low recall region, as only a small number n of instances are labeled there. We stress the fact that we use the same labeled set for all the different algorithms and features, meaning that the relative position of the curves is by itself significant.

A few high-level trends emerge when looking at the plots. First is that in all cases, all our classifiers that use complexity features outperform the labels from the reference data provider. This is a notable result because it means that it is possible to use a training set labeled by another algorithm to train a classifier that can outperform the original algorithm. This observation also makes sense because the complexity features help model a key characteristic of spam: the repetitive, uninformative nature of the posted content associated with a user/IP/embedded hostname.

Second, from Figures 3.5a, 3.5c, and 3.5d, we see that for most feature sets the Latent LR outperform the Plain LR. Thus, a noise-tolerant algorithm like Latent LR can provide an improvement over Plain LR, especially in a data set that could contain noisy labels. The exception is Figure 3.5b, where the performance of Plain and Latent LR are indistinguishable. This happens in Figure 3.5b because in a quadratic expansion with a higher number of dimensions, the adverse affects of mislabeling are less noticeable because the decision boundary itself is more complex, i.e., more nonlinear with respect to the original, unexpanded features. Still, because Latent LR performs better than Plain LR in most cases and in the worst case performs at least as well as Plain LR, it is safe to use the Latent LR in practice.

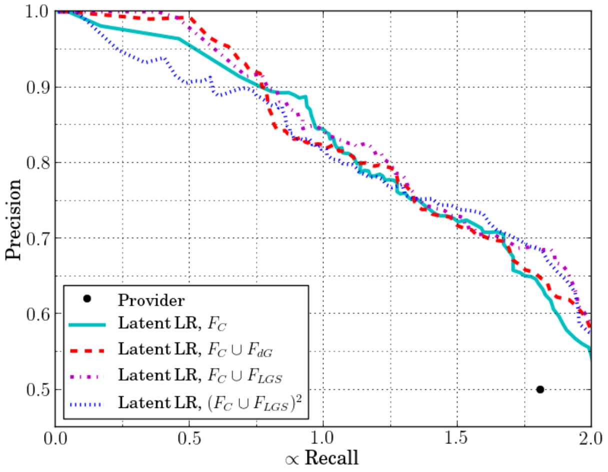


Figure 3.6: A side-by-side comparison of the precision-recall performance for different feature sets using Latent LR.

Given the Latent LR algorithm, which of the feature set combinations are most appropriate for deploying in practice? As we can see from Figure 3.6, the performance of the different feature sets are relatively close overall. A slight edge may be awarded to the $F_C \cup F_{LGS}$ feature set because it has good performance in both the high-precision region and high-recall regions of the precision-recall space, especially in the parts of the graph where unnormalized recall is ≤ 0.75 and ≥ 1.75, respectively. By contrast, other feature sets may favor higher recall at the expense of lower precision like $(F_C \cup F_{LGS})^2$, while yet others may favor high precision at the expense of recall. For example, at a recall of 0.5, $F_C \cup F_{LGS}$ scores a precision of over 97%, whereas the quadratic expansion $(F_C \cup F_{LGS})^2$ only achieves 90% precision. And at a recall of 1.8, $F_C \cup F_{LGS}$ achieves a precision of 69%, whereas the precision for F_C and $F_C \cup F_{dG}$ is 65%.

On a computational complexity standpoint, there are three phases with distinct performances: features extraction, learning, and scoring. The scoring phase consists solely of an inner product of size the number of features and a scalar comparison, thus its running time is negligible. Depending on the number of features used, training a plain logistic regression on the whole dataset takes from a few minutes to an hour on a 2 GHz Intel Core i7. On the same machine, training a latent logistic regression takes between half an hour to several hours. The feature extraction phase is the most computationally expensive and is done on a Hadoop grid with eight mappers and six reducers. Grouping comments in the map and sort phases and compressing groups in the reduce phase takes a few hours, thus computing the four core groupings takes about half a day.

Overall, because of the performance of the complexity features combined with the latent variable model, they provide a promising complementary to existing comment spam detection techniques.

Finally, there are two arbitrarily fixed parameters we did not evaluate. The first is the Δt parameter we use for IP anti-aliasing. Everything else considered equal, decreasing this parameter breaks down groupings into smaller ones, until all IP groupings are singletons. On the contrary, increasing it will merge all groups sharing the same IP together, thus loosing the anti-aliasing benefit. Thus, there is arguably an optimal value for the parameter, which we did not try to evaluate. An adversary might also want to arrange for posting her messages with a temporal rate smaller than Δt^{-1}, thus it also acts as an implicit upper bound on the spam rate of a given IP.

In the same fashion, the second parameter which evaluation we did not take is the time window on which we compute the features. This parameter is equal to two months in our evaluation, and presents a behavior similar to the one of Δt when varied.

3.6 Open Problems

This chapter described a robust approach of detecting spam in social media. Our approach uses content complexity of comments with a latent logistic regression classifier. This approach is a first step in detecting comment spam from noisy and missing labels using language-agnostic features and potentially evasion-resistant technique. Our evaluation shows the approach is general and robust, and provides new insights for industry to further improve their techniques for fighting comment spam.

Moving forward, a diverse of research directions can be pursued to improve the performance of the proposed comment spam detection system.

Adapted Evasion Strategies

Although our evaluation is temporally-consistent and uses a large scale, real dataset containing evading adversaries, we note that this is unfortunately not fully indicative of the real-life performance of our technique. Indeed, if the system were fielded, we would expect the adversaries to react to its specific presence by crafting adapted evasions. Under its current state, our proposal might be vulnerable to a series of targeted evasion strategies.

One such strategy would be to artificially inflate the content complexity scores by appending random characters to each messages. For such an attack, a defense technique would be to develop a method to detect those random character runs, use their presence and length as a new feature, and discard them before computing the compressibility of the messages. We believe that character-level Markovian techniques such as Hidden Markov Models (Baum and Petrie 1966) or Prediction by Partial Matching (Cleary and Witten 1984) would be well suited for this purpose.

Alternatively, a stronger attacker might increase the content complexity by appending natural language content that is either synthetically generated, or copied from an offline (e-books) or online (other blog content or comment) source. In this situation, we can instead focus on the recurring parts in each message. In order to detect significant portions of repeated content in an efficient and scalable manner, we can leverage text fingerprinting (Hoad and Zobel 2003) and min-hashing (Broder et al. 1998) techniques for instance.

In addition to the content-complexity based features, there are a variety of other features we can incorporate to increase evasion robustness. Among them, the containment of spam trigger words and URLs has been shown to be very discriminative (Xie et al. 2008), as well as the posting behavior of users (Ramachandran et al. 2007), user reputation (B.-C. Chen et al. 2011) and user-comment relationship graph (Mishra and Rastogi 2012).

Online Update and Operation

In production, spam detectors are usually deployed and operated in an online setting. When a new example arrives, features need to be extracted from the data and fed to the classifier. The classifier makes a prediction, is told if its prediction is correct, and updates its model accordingly. While our classifier itself can be easily adapted for online setting, the feature sets are very intertwined: when a new data point is assigned to groups with several messages in it, the complete calculation of the content complexity measure has to be re-triggered and back-propagated to all affected messages. This could lead to an expensive online update step for several data points. We leave it as a future work to develop an efficient (incremental) approach for extracting content complexity features from comment data.

Part II

Hardening Machine Learning

Chapter 4

Convex Polytope Machine

4.1 Introduction

In this chapter, we take a systematic approach to evasion on detection pipelines that is independent of the specifics of a given application domain and focuses on the intrinsic properties of the machine learning classifier. To this end, we simplify the problem of modeling the evasions in the original observation space Ω by instead working in the mapped feature space $\mathbb{R}^d$. This amounts to identifying Ω to $\mathbb{R}^d$ and ψ to the identity application.

Chapter 3 presented a novel feature extractor ψ in the context of comment spam detection. The temporally consistent evaluation of the resulting detection pipeline provided an indirect evidence of its robustness to evasion. In contrast, demonstrating that a strongly motivated adversary can not reasonably evade detection would constitute a direct evidence of evasion robustness.

Unfortunately, modeling the adversary directly in the observation space Ω and constructing corresponding evasion algorithms is a difficult task. In particular, estimating the cost incurred by the adversary for changing a spam comment ω into an evading one ω' requires estimating in turn the importance of many hard to measure factors. For instance, we would need to quantitatively understand how adding or modifying the intended content in ω impacts the effectiveness of the message on end-users. Likewise, we would also need to understand how expensive it is to create an evading observation ω'. This quantity decomposes into at least the computational cost and the hardware, electricity and network communication costs.

As mentioned in Section 2.2, we can use the natural L_ρ distances for $\rho \in \{0, 1, 2, \infty\}$ to quantify the minimal amount of distortion between a correctly recognized input x and a corresponding evading input x'. In the optimal evasion framework, problem (2.2), which we reproduce here,

$$\underset{x' \in \mathbb{R}^d}{\text{minimize}}\ \Delta(x, x') \quad \text{subject to } c(x) \neq c(x')$$

serves as the evaluation metric for evasion robustness. According to this metric, the harder it is to evade classifier c, the larger the optimal value of problem (2.2) is.

Optimal Evasion and Large Margin Learning

An interesting interpretation of the optimal evasion problem is that a hard-to-evade classifier c is also more stable in its decisions. That is, there exists some large stability radius $r > 0$ such that for any instance x of interest, it is the case that any other instance x' at distance less than r from x gets assigned the same decision $c(x)$. Figure 4.1 illustrates this observation in the case of the Euclidean L_2 distance.

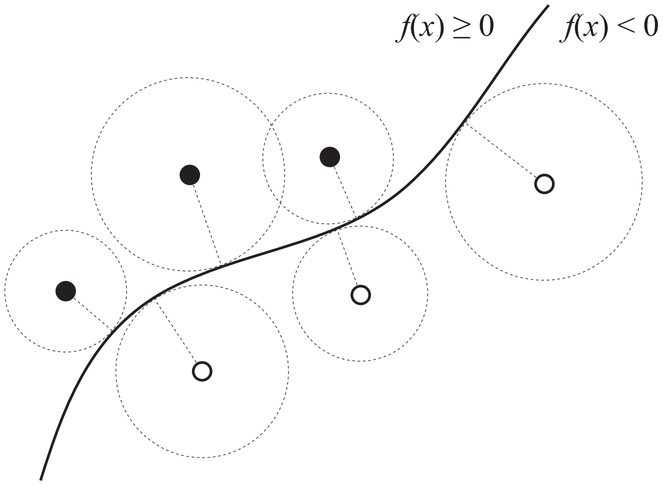


Figure 4.1: Evasion, stability and large-margin classification are related concepts. The decision boundary of a classifier f is shown in solid black together with six correctly classified instances (three on each side). For each instance is shown the corresponding "disk of decision stability" where any other point is also correctly classified by f. The larger the disks are, the more evasion resistant f is.

The correctly classified positive and negative instances can freely wiggle inside their corresponding bounding circles without modifying the output of the classifier. Because the radius of a bounding circle is effectively the distance of the instance to the decision boundary separating the two classes, maximizing the stability regions corresponds to maximizing the *separation margins*, or distances to the decision boundary, that the classifier induces over the dataset. Evasion robustness is hence closely related to the *large-margin* understanding of classifiers (Smola et al. 2000), with the important caveat that evasions occur at testing time, whereas large-margin is only applied at training time.

Large-margin learning methods aim at finding the model $f \in \mathcal{F}$ which maximizes the distances of the training points to the decision boundary in some sense. A historically important example of learning algorithm that implements the large-margin idea is the Support Vector Machine. To simplify, we can assume there exists a classifier in $\mathcal{F}$ which perfectly separates the training set. That is, a classifier f for which $f(x^{(i)}) = 1$ if and only if $y_i = 1$ over the training set. Support Vector Machines find a perfectly separating classifier such that the minimal distance of any instance to the separation boundary is maximal. However, our current understanding of the good practical and theoretical performance of Support Vector Machines goes beyond this geometrically separable case. Indeed, proper generalizations of a separation margin exist for non-separable data which make the large-margin method essentially collapse to ERM (Bartlett et al. 2003).

The Convex Polytope Machine

In this chapter, we exploit the relationship between large-margin learning and evasion robustness. Our hope is that by grounding our classifier in a large-margin framework, our method inherits favorable evasion resistant properties. We also aim at creating a generic-purpose machine learning technique with outstanding training time performance on large training sets and competitive classification accuracy and testing time performance. Part of the work presented below was first reported in (Kantchelian et al. 2014).

Many application domains of machine learning use massive data sets in dense medium-dimensional or sparse high-dimensional spaces. Some domains also require near real-time responses in both the prediction and the model training phases. These applications often deal with inherent non-stationarity; thus, the models need to be constantly updated in order to catch up with drift. Today, the de facto model class for tasks at these scales is a linear classifier which is trained under the machinery of linear Support Vector Machine. Indeed, since (Shalev-Shwartz et al. 2007) demonstrated both theoretically and experimentally that large margin linear classifiers can be efficiently trained at scale using stochastic gradient descent, their Pegasos algorithm has become a standard tool for the machine learning practitioner.

As the learning capacity of the linear model class increases unboundedly with the number of dimensions d, a linear classifier in very high dimension d is expected to have a considerable expressiveness power. This argument is often understood as "everything is separable in high dimensional spaces; hence, linear separation is good enough". However, in practice, deployed systems rarely use a single naked linear separator. One explanation for this gap between theory and practice is that while a single hyperplane might indeed perfectly separating both classes in very high dimensions, the resulting separation margin might be very small. Since the classifier margin also accounts for the generalization power and evasion robustness, we might experience poor future classification performance in this scenario.

Figure 4.2 provides a two-dimensional example of a data set that has a small margin when using a single separator (solid line) despite being linearly separable and intuitively easily classified. The intuition that the data is easily classified comes from the data naturally separating into three clusters with two of them in the positive class. Such clusters can form due to the positive instances being generated by a collection of different processes.

As Figure 4.2 shows, a way of increasing the margins is to introduce two linear separators (dashed lines), one for each positive cluster. We take advantage of this intuition to design a novel machine learning algorithm that will provide larger margins than a single linear classifier while still enjoying much of the computational effectiveness of a simple linear separator. Our algorithm learns a bounded number of linear classifiers simultaneously. The global classifier aggregates all the sub-classifiers decisions by taking the maximum sub-classifier score. The maximum aggregation has

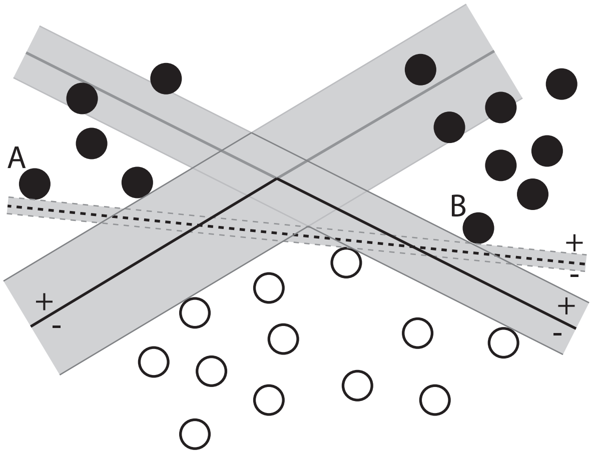


Figure 4.2: Positive (•) and negative (◦) instances in continuous two dimensional feature space. Instances are perfectly linearly separable (dashed line), although with small margin due to positive instances (A & B) having conflicting patterns. We can obtain higher margin by separately training two linear sub-classifiers (solid lines) on left and right clusters of positive instances, each against all the negative instances, yielding a prediction value of the maximum of the sub-classifiers.

the effect of assigning a positive point to a unique sub-classifier. Geometrically, the corresponding model class is the class of convex polytope separators in $\mathbb{R}^d$.

In this chapter, we propose a novel algorithm for learning convex polytope classifiers with superior empirical performance to existing algorithms. We call our algorithm the Convex Polytope Machine, or CPM for short. Our experimental evaluations of the CPM on large-scale data sets from distinct domains (MNIST handwritten digit recognition, text topic, and web security) demonstrate that the CPM trains models faster, sometimes by several orders of magnitude, than state-of-the-art similar approaches and kernel methods while achieving comparable or better classification performance. Our empirical results suggest that, unlike prior similar approaches, we do not need to control the number of sub-classifiers (sides of the polytope) to avoid overfitting.

Related Work

Fischer (1995) focuses on finding the polygon with the fewest misclassified points drawn independently from an unknown distribution using an algorithm with a running time of more than $O(n^{12})$ where n is the number of sample points. We instead focus on finding good, not optimal, polygons that generalize well in practice despite having fast running times. Our focus on generalization leads us to maximize the margin, unlike this work, which actually minimizes it to make their proofs easier.

Takacs (2010) proposes algorithms for training convex polytope classifiers based on the smooth approximation of the maximum function. While his algorithms use smooth approximation during training, it uses the original formula during prediction, which introduces a gap that could deteriorate the accuracy. The proposed algorithms achieve similar classification accuracy to several nonlinear classifiers, including nearest neighbors, decision tree and kernel support vector machines. However, the training time of the algorithms is often much longer than those nonlinear classifiers, sometimes by an order of magnitude than the ID3 algorithm and eight times longer than kernel support vector machines on medium size datasets, diminishing the motivation to use the proposed algorithms in realistic settings.

The authors of (Wang et al. 2011) propose an Adaptive Multi-hyperplane Machine (AMM) algorithm that is fast during both training and prediction, and capable of handling nonlinear classification problems. They develop an iterative algorithm based on stochastic gradient descent to search for the number of hyperplanes and train the model. Their experiments on several large data sets show that AMM is nearly as fast as the state-of-the-art linear SVM solver and achieves classification accuracy between linear and kernel support vector machines. In contrast to the theoretical analysis of AMM, our experiments and to some extent our theoretical analysis suggest that increasing the number of hyperplanes, or polytope faces, does not significantly decrease generalization power. Building on the stochastic gradient descent training algorithm, we introduce also an important corrective procedure to increase the utilization of hyperplanes. Finally, we demonstrate how the AMM model can be derived under the more fundamental margin optimization framework.

Manwani and Sastry (2010) propose two methods for learning polytope classifiers, one based on the logistic function, and another based on the perceptron method (Manwani and Sastry 2013), and propose alternating optimization algorithms to train the classifiers. However, they only evaluate the proposed methods on a few small data sets (with no more than 1000 samples in each), and do not compare them to other widely used non-linear classifiers. It is unclear how applicable these algorithms are to large-scale data. Our work makes three significant contributions over their work. First, we derive the formulation from a large-margin argument and obtaining a regularization term that is missing in (Manwani and Sastry 2013). Second, we show we can safely restrict the choice of assignments to only positive instances, leading to a training time optimization heuristic. Third, we demonstrate higher performance on non-synthetic, large scale data sets when using two convex polytope classifiers together.

Finally, we note that the CPM is a special case of the more general latent support vector machine formulation (Felzenszwalb et al. 2010). In particular, the latent variable represents the sub-classifier, or face of the polytope, used for classifying the given instance.

4.2 Large-Margin Convex Polytopes

In this section, we derive and discuss several alternative optimization problems for finding a large-margin convex polytope which separates binary labeled points of $\mathbb{R}^d$.

Problem Setup and Model Space

Let $\mathcal{D} = \{(x^{(i)}, y_i)\}_{1\leq i\leq n}$ be a binary labeled data set of n instances, where $x^{(i)} \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$. For the sake of notational brevity, we assume that an instance $x^{(i)}$ includes a constant unitary component corresponding to a bias term. Our prediction problem is to find a classifier $c : \mathbb{R}^d \to \{-1, 1\}$ such that $c(x^{(i)})$ is a good estimator of y_i. To do so, we consider classifiers constructed from convex K-faced polytope separators for a fixed positive integer K. Let $\mathcal{P}_K$ be the model space of convex K-faced polytope separators:

$$\mathcal{P}_K = \left\{ f : \mathbb{R}^d \to \mathbb{R} \,\middle|\, f(x) = \max_{1 \le k \le K} (Wx)_k, W \in \mathbb{R}^{K \times d} \right\}$$

For each such function f in $\mathcal{P}_K$, we can get a classifier c_f such that $c_f(x)$ is 1 if $f(x) > 0$ and -1 otherwise. This model space corresponds to a shallow single hidden layer neural network with a max aggregator.

Note that when $K = 1$, $\mathcal{P}_1$ is simply the space of all linear classifiers. Importantly, when $K \ge 2$, elements of $\mathcal{P}_K$ are not guaranteed to have additive inverses in $\mathcal{P}_K$. Thus, the labels $y = -1$ and $y = +1$ are not interchangeable. Geometrically, the negative class remains enclosed within the convex polytope while the positive class lives outside of it, leading to the label asymmetry.

To construct a classifier without label asymmetry, we can use two polytopes, one with the negative instances on the inside the polytope to get a classification function f_- and one with the positive instances on the inside to get f_+. From these two polytopes, we construct the classifier c_{f_-,f_+} where $c_{f_-,f_+}(x)$ is 1 if $f_-(x) - f_+(x) \ge 0$ and -1 otherwise. Both the evaluation section and Chapter 6 use this double convex polytope construction. Figure 4.3 illustrates the resulting model class.

To better understand the nature of the faces of a single polytope, for a given polytope W and a data point x, we denote by $z_W(x)$ the index of the maximum sub-classifier for x:

$$z_W(x) \;=\; \operatorname*{argmax}_{1 \le k \le K} (Wx)_k$$

We call $z_W(x)$ the *assigned sub-classifier* for instance x. When clear from context, we drop W from z_W. We use the notation W_k to designate the k-th row of matrix W, which corresponds to the k-th face of the polytope, or the k-th sub-classifier. Hence, $W_{z(x)}$ identifies the weights of the separator assigned to x.

We now pursue a geometric large-margin-based approach for formulating the concrete optimization problem. To simplify notation and without loss of generality, we suppose that W is row-normalized such that $\|W_k\| = 1$ for all k. We also initially suppose our data set is perfectly separable by a K-faced convex polytope.

Margins for Convex Polytopes

When $K = 1$, the problem reduces to finding a good linear classifier and only a single natural margin δ of the separator exists (Cortes and Vapnik 1995):

$$\delta_W \;=\; \min_{1 \le i \le n} y_i W_1 x^{(i)}$$

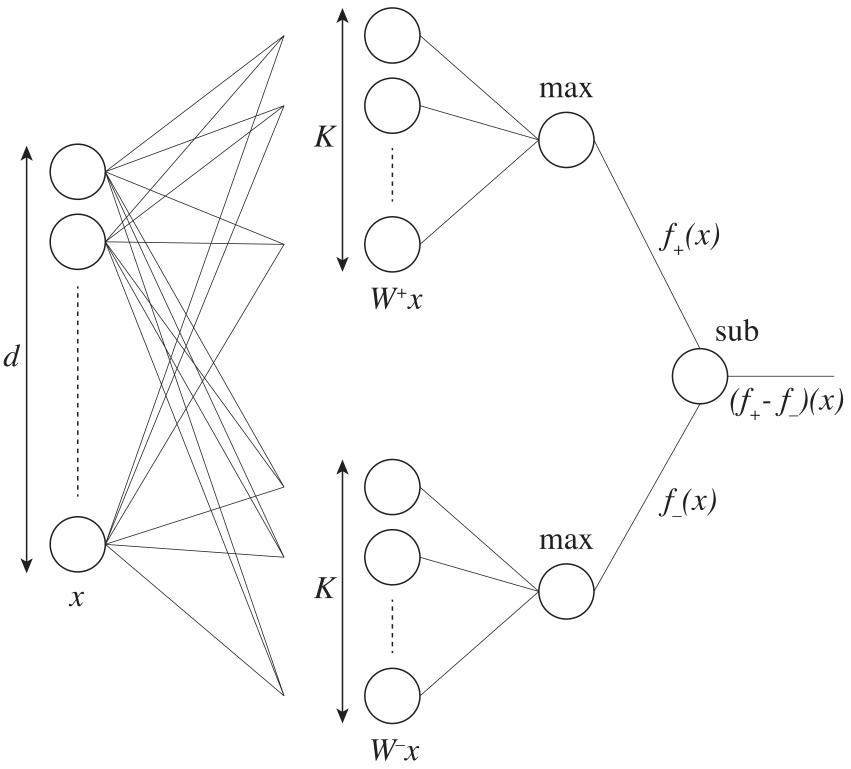


Figure 4.3: Neural network representation of the difference of two convex polytope machines. For both CPMs, the ouputs of a hidden layer of K independent linear sub-classifiers are max-ensembled.

Maximizing δ_W yields the well known linear support vector machine. However, multiple notions of margin exist for a K-faced convex polytope with $K \geq 2$. We distinguish three notions of margin in what follows.

Let the *worst case margin* δ_W^{WC} be the smallest margin of any point to the polytope. Over all the K sub-classifiers, we find the one with the minimal margin to the closest point assigned to it:

$$\begin{aligned}\delta_W^{\mathrm{WC}} &= \min_{1\leq i\leq n} y_i W_{z(x^{(i)})} x^{(i)} \\ &= \min_{1\leq k\leq K} \min_{i:z(x^{(i)})=k} y_i W_k x^{(i)}\end{aligned}$$

The worst case margin is very similar to the linear classifier margin but suffers from an important drawback. Maximizing δ^{WC} leaves $K-1$ sub-classifiers wiggling while over-focusing on the sub-classifier with the smallest margin. Figure 4.4 presents a geometrical intuition for this fact.

Thus, we instead focus on the *total margin*, which measures each sub-classifier's margin with

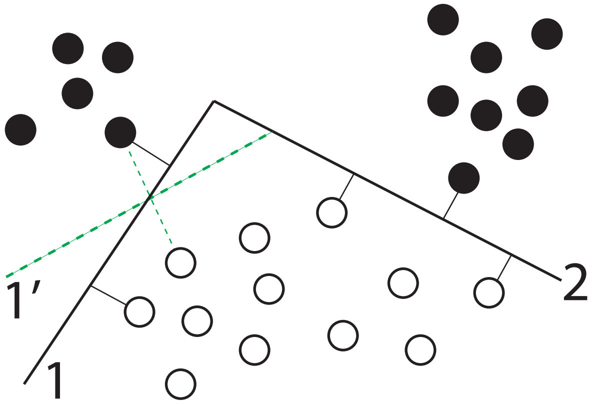


Figure 4.4: The worst-case margin is insensitive to wiggling of sub-classifiers having non-minimal margin. Sub-classifier 2 has the smallest margin, and sub-classifier one is allowed to freely move without affecting δ^{WC}. For comparison, the largest-margin solution $1'$ is shown (dashed lines).

respect to just its assigned points. The total margin δ_W^{T} is the sum of the K sub-classifiers margins:

$$\delta_W^{\mathrm{T}} = \sum_{k=1}^{K} \min_{i:z(x^{(i)})=k} y_i W_k x^{(i)}$$

The total margin gives the same importance to each of the K sub-classifier margins.

A potential improvement over the total margin is to give more importance to sub-classifiers which are responsible for classifying a larger portion of the data set. We can accomplish this by weighting each sub-classifier k by its number of classified instances $\alpha_k = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{z(x^{(i)})=k}$. We have:

$$\delta_W^{\mathrm{E}} = \sum_{k=1}^{K} \alpha_k \min_{i:z(x^{(i)})=k} y_i W_k x^{(i)} \tag{4.1}$$

We call δ_W^E the *effective margin*.

Maximizing the Margin

We now turn to the question of maximizing the margin. We show the step-by-step derivation a smoothed but non-convex optimization problem for maximizing the total margin.

$$\max_W \delta_W^{\mathrm{T}} \tag{4.2}$$
$$\text{s.t. } \|W_1\| = \cdots = \|W_K\| = 1$$

Introducing one additional variable ζ_k per classifier, problem (4.2) is equivalent to:

$$\begin{aligned} \max_{\mathbf{W},\zeta} \sum_{k=1}^{K} \zeta_k & \qquad (4.3) \\ \text{s.t. } \forall i, \zeta_{z(x^{(i)})} \le y_i W_{z(x^{(i)})} x^{(i)} & \\ \zeta_1 > 0, \ldots, \zeta_K > 0 & \\ \|W_1\| = \cdots = \|W_K\| = 1 & \end{aligned}$$

Considering the unnormalized rows W_k/ζ_k, we obtain the following equivalent formulation:

$$\max_{W} \sum_{k=1}^{K} \frac{1}{\|W_k\|} \qquad (4.4)$$

$$\text{s.t. } \forall i, 1 \le y_i W_{z(x^{(i)})} x^{(i)} \qquad (4.5)$$

When $y = -1$, $z(x^{(i)})$ satisfying the margin constraint (4.5) implies that the constraint holds for every sub-classifier k since $y_i W_k x^{(i)}$ is minimal at $k = z(x^{(i)})$. Thus, when $y = -1$, we can enforce the constraint for all k yielding the following equivalent problem:

$$\begin{aligned} \max_{W} \sum_{k=1}^{K} \frac{1}{\|W_k\|} & \qquad (4.6) \\ \text{s.t. } \forall i : y_i = -1, \forall k \in \{1, \ldots, K\}, 1 + W_k x^{(i)} \le 0 & \\ \forall i : y_i = +1, 1 - W_{z(x^{(i)})} x^{(i)} \le 0 & \end{aligned}$$

Finally, we can relax the objective into a convex one by minimizing the sum of the inverse squares of the terms instead of maximizing the sum of the terms. We obtain the following smoothed problem:

$$\min_{W} \sum_{k=1}^{K} \|W_k\|^2 \qquad (4.7)$$

$$\text{s.t. } \forall i : y_i = -1, \forall k \in \{1, \ldots, K\}, 1 + W_k x^{(i)} \le 0 \qquad (4.8)$$

$$\forall i : y_i = +1, 1 - W_{z(x^{(i)})} x^{(i)} \le 0 \qquad (4.9)$$

The objective (4.7) is now the familiar convex L_2 regularization term $\|W\|^2$. The negative samples constraints (4.8) are convex (linear functions), but the positive terms (4.9) result in non-convex constraints because of the instance-dependent assignment z. As for the Support Vector Machine, we can introduce n slack variables ξ_i and a regularization factor $C > 0$ for the common case of noisy, non-separable data. Hence, the practical problem becomes:

$$\begin{aligned} &\min_{W,\xi} \|W\|^2 + C\sum_{i=1}^{n} \xi_i \\ &\text{s.t. } \forall i : y_i = -1, \forall k \in \{1,\ldots,K\}, 1 + W_k x^{(i)} \le \xi_i \\ &\qquad \forall i : y_i = +1, 1 - W_{z(x^{(i)})} x^{(i)} \le \xi_i \\ &\qquad \forall i, \xi_i \ge 0 \end{aligned} \tag{4.10}$$

Following the same steps, we obtain the following problem for maximizing the worst-case margin. The only difference is the regularization term in the objective function which becomes $\max_k \|W_k\|^2$ instead of $\|W\|^2$.

The goal of our relaxation is to demonstrate that our solution involves two intuitive steps: *(i)* assigning positive instances to sub-classifiers, and *(ii)* solving a collection of SVM-like sub-problems. While our solution taken as a whole remains non-convex, this decomposition isolates the non-convexity to a single intuitive assignment problem that is similar to clustering. This isolation enables us to use intuitive heuristics or clustering-like algorithms to handle the non-convexity. Indeed, in our final form (4.10), if the optimal assignment function $z(x^{(i)})$ of positive instances to sub-classifiers were known and fixed, the problem would be reduced to a collection of perfectly independent convex minimization problems. Each such sub-problem corresponds to a classical SVM defined on all negative instances and the subset of positive instances assigned by $z(x^{(i)})$.

Choice of K, Generalization Bound for CPM

Assuming we can efficiently solve this optimization problem, we would need to adjust the number K of faces and the degree C of regularization. The following result gives a generalization bound for the CPM. For $B_1, \ldots, B_k \ge 0$, let $\mathcal{F}_{K,B}$ be the following subset of the set $\mathcal{P}_K$ of convex polytope separators:

$$\mathcal{F}_{K,B} = \left\{ f : \mathbb{R}^d \to \mathbb{R} \,\middle|\, f(x) = \max_{1\le k\le K} (Wx)_k, W \in \mathbb{R}^{K\times d}, \forall k, \|W_k\| \le B_k \right\}$$

Theorem 1. *There exists some constant* $A > 0$ *such that for all distributions* P *over* $\mathbb{R}^d \times \{-1, 1\}$, K *in* $\{1, 2, 3, \ldots\}$, $B_1, \ldots, B_k \ge 0$, *and* $\delta > 0$, *with probability at least* $1 - \delta$ *over the training set* $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \sim P$, *any* f *in* $\mathcal{F}_{K,B}$ *is such that:*

$$P(yf(x) \le 0) \le \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i f(x^{(i)})) + A\frac{\sum_k B_k}{\sqrt{n}} + \sqrt{\frac{\ln(2/\delta)}{2n}}$$

This is a uniform bound on the 0-1 risk of classifiers in $\mathcal{F}_{K,B}$. It shows that with high probability, the risk is bounded by the empirical hinge loss plus a capacity term that decreases in $n^{-1/2}$ and is proportional to the sum of the sub-classifier norms. Note that as we have $\sum_k \|W_k\| \le \sqrt{K}\|W\|$, the capacity term is essentially equivalent to $\sqrt{K}\|W\|$. As a comparison, the generalization error for AMM (Wang et al. 2011), a related piecewise-linear classification method has previously been shown to be proportional to $K\|W\|$ in (Wang et al. 2011, Thm. 2). In practice, this bound is very loose as it does not explain the observed absence of over fitting as K gets large. We experimentally demonstrate this phenomenon in the evaluation section. We conjecture that there exists a bound that is independent of K altogether. The proof of Theorem 1 relies on a result of (Bartlett and Mendelson 2003) on Rademacher complexities. We first prove that the Rademacher complexity of $F_{K,B}$ is in $O(\sum_k B_k/\sqrt{n})$. We then invoke Theorem 7 in (Bartlett and Mendelson 2003) to show our result. The complete proof is as follows.

Proof. The Rademacher complexity of $\mathcal{F}_{K,B}$ is defined as

$$\mathcal{R}_n(\mathcal{F}_{K,B}) = \mathbb{E}_{(x^{(i)})}\mathbb{E}_{\epsilon}\left[\sup_{f\in\mathcal{F}_{K,B}}\left|\frac{1}{n}\sum_i \epsilon_i f(x^{(i)})\right|\right]$$

where the ϵ_i are ±1 i.i.d. Bernoulli with probability 1/2.

It is also possible to define the Gaussian Rademacher complexity of $\mathcal{F}_{K,B}$ as:

$$\mathcal{G}_n(\mathcal{F}_{K,B}) = \mathbb{E}_{(x^{(i)})}\mathbb{E}_{g}\left[\sup_{f\in\mathcal{F}_{K,B}}\left|\frac{1}{n}\sum_i g_i f(x^{(i)})\right|\right]$$

where the g_is are i.i.d. standard normal variables.

By Lemma 4 in (Bartlett and Mendelson 2003), there exists an absolute constant b such that for every $\mathcal{F}_{K,B}$ and n we have $\mathcal{R}_n(\mathcal{F}_{K,B}) \leq b\mathcal{G}_n(\mathcal{F}_{K,B})$. Thus, we can provide a bound on the Gaussian Rademacher complexity. In our case, this can be directly done by invoking Theorem 14 in (Bartlett and Mendelson 2003). Indeed, $q_1, \ldots, q_k \mapsto \max(q_1, \ldots, q_k)$ is a Lipchitz function with constant 1, thus $\mathcal{F}_{K,B}$ can be viewed as the composition of the max function with the real valued classes of linear separators $\mathcal{F}_i$ that are such that

$$\mathcal{F}_i = \{x \mapsto w^{\intercal}x \mid \|w\| \leq B_i\}$$

So we have that $\mathcal{G}_n(\mathcal{F}_{K,B}) \leq 2\sum_{k=1}^{K}\mathcal{G}_n(\mathcal{F}_k)$. The Gaussian Rademacher complexities of each of these $\mathcal{F}_k$s is bounded by $B_k/\sqrt{n}$ by a standard argument as follows:

$$\begin{aligned}
\mathcal{G}_n(\mathcal{F}_k) &= \mathbb{E}_{(x^{(i)})}\mathbb{E}_g\left[\sup_{\|w\|\le B_k}\left|\frac{1}{n}\sum_i g_i w^\intercal x^{(i)}\right|\right]\\
&= \mathbb{E}_{(x^{(i)})}\mathbb{E}_g\left[\sup_{\|w\|\le B_k}\frac{1}{n}w^\intercal\left(\sum_{i=1}^n x^{(i)}g_i\right)\right]\\
&= \mathbb{E}_{(x^{(i)})}\mathbb{E}_g\frac{B_k}{n}\|\sum_{i=1}^n x^{(i)}g_i\|\\
&\le \mathbb{E}_{(x^{(i)})}\frac{B_k}{n}\sqrt{\mathbb{E}_g\|\sum_{i=1}^n x^{(i)}g_i\|^2}\\
&= \mathbb{E}_{(x^{(i)})}\frac{B_k}{n}\sqrt{\sum_{i=1}^n\|x^{(i)}\|^2}\\
&\le \frac{B_k}{\sqrt{n}}
\end{aligned}$$

Hence, there exists a universal constant $A > 0$ such that

$$\mathcal{R}_n(\mathcal{F}_{K,B}) \le A\sum_k \mathcal{G}_n(\mathcal{F}_k) = A\frac{\sum_k B_k}{\sqrt{n}}$$

Finally, we apply Theorem 7 of (Bartlett and Mendelson 2003) where ϕ is taken to be the hinge loss, and obtain the desired result. □

4.3 SGD-based Learning

In this section, we present a learning algorithm based on Stochastic Gradient Descent (SGD) for approximately solving the total margin maximization problem (4.10). The choice of SGD is motivated by two factors. First, we would like our learning technique to efficiently scale to several million instances of sparse high dimensional space. The sample-iterative nature of SGD makes it a very suitable candidate for this end (Bottou 2010). Second, the optimization problem we are solving is non-convex. SGD has recently been shown to work well for such learning problems when near-optimum solutions are acceptable (Hinton 2012).

Problem (4.10) can be expressed as an unconstrained minimization problem as follows:

$$\min_W \sum_{i:y_i=-1}\sum_{k=1}^K [1 + W_k x^{(i)}]_+ + \sum_{i:y_i=+1}[1 - W_{z(x^{(i)})}x^{(i)}]_+ + \lambda\|W\|^2$$

where $[x]_+ = \max(0, x)$ and $\lambda > 0$. This form reveals the strong similarity with optimizing K unconstrained linear SVMs (Shalev-Shwartz et al. 2007). The difference is that although each sub-classifier is trained on all the negative instances, positive instances are associated to a unique sub-classifier. From the unconstrained form, we can derive the stochastic gradient descent Algorithm 1.

For the positive instances, we isolate the task of finding the assigned sub-classifier z to a separate procedure ASSIGN. We use the Pegasos inverse schedule $\eta_t = 1/(\lambda t)$.

Algorithm 1 Stochastic gradient descent algorithm for solving problem (4.10).

```
function SGDTRAIN($\mathcal{D}, \lambda, T, (\eta_t), h$)
    Initialize $W \in \mathbb{R}^{K \times d}, W \leftarrow \mathbf{0}$

    for $t \leftarrow 1, \ldots, T$ do
        Pick $(x, y) \in \mathcal{D}$

        if $y = -1$ then
            for $k \leftarrow 1, \ldots, K$ do
                if $W_k x > -1$ then
                    $W_k \leftarrow W_k - \eta_t x$
        else if $y = +1$ then
            $z \leftarrow \operatorname{argmax}_k W_k x$
            if $W_z x < 1$ then
                $z \leftarrow$ ASSIGN($W, x, h$)
                $W_z \leftarrow W_z + \eta_t x$

        $W \leftarrow (1 - \eta_t \lambda) W$
    return $W$
```

Assignment Heuristic

Since the optimization problem (4.10) is non-convex, a pure SGD approach could get stuck in a low-quality local optimum and we, indeed, found that this problem occurs in practice. These optima assign most of the positive instances to a small number of sub-classifiers. In this configuration, the remaining sub-classifiers serve no purpose. Intuitively, the algorithm clustered the data into large "super-clusters" ignoring the more subtle sub-clusters comprising the larger super-clusters. The large clusters represent an appealing local optima since breaking one down into sub-clusters often requires transitioning through a patch of lower accuracy as the sub-classifiers align themselves to the new cluster boundaries. We view the local optima as the algorithm underfitting the data by using too simple of a model. In this case, the algorithm needs encouragement to explore more complex clusterings.

With this intuition in mind, we add a term encouraging the algorithm to explore higher diversity configurations of the sub-classifiers. To do so, we use the entropy of the random variable $Z = \operatorname{argmax}_k W_k x$ where $x \sim \mathcal{D}^+$ is a distribution defined on the set of all positive instances as follows. Let n_k be the number of positive instances assigned to sub-classifier k, and n be the total number of positive instances. We define $\mathcal{D}^+$ as the empirical distribution on $\left(\frac{n_1}{n}, \frac{n_2}{n}, \ldots, \frac{n_k}{n}\right)$. The entropy is zero when the same classifier fires for all positive instances, and maximal at $\log_2 K$ when every classifier fires on a K^{-1} fraction of the positive instances. Thus, maximizing the entropy encourages the algorithm to break down large clusters into smaller clusters of near equal size.

We use this notion of entropy in our heuristic procedure for assignment, described in Algorithm 2. ASSIGN takes a predefined minimum entropy level $h \geq 0$ and compensates for disparities in how positive instances are assigned to sub-classifiers, where the disparity is measured by entropy. When the entropy is above h, ASSIGN uses the natural $\operatorname{argmax}_k W_k x$ assignment. Conversely, if the current entropy is below h, then it picks an assignment that is guaranteed to increase the entropy. Thus, when $h = 0$, there is no adjustment made. It keeps a dictionary UNADJ mapping the previous points it has encountered to the unadjusted assignment that the natural argmax assignment would had made at the time of encountering the point. We write UNADJ $+ (x, k)$ to denote the new dictionary U such that $U[v]$ is equal to k if $v = x$ and to UNADJ$[v]$ otherwise. Dictionary UNADJ keeps track of the assigned positives per sub-classifiers, and serves to estimate the current entropy in the configuration without needing to recompute every prior point's assignment.

Algorithm 2 Heuristic maximum assignment algorithm. The input is the current weight matrix W, positive instance x, and the desired assignment entropy $h \geq 0$.

```
Initialize UNADJ← {}
function ASSIGN($W, x, h$)
    $k_{\text{unadj}} \leftarrow \operatorname{argmax}_k W_k x$

    if ENTROPY(UNADJ $+ (x, k_{\text{unadj}})) \geq h$ then
        $k_{\text{adj}} \leftarrow k_{\text{unadj}}$
    else
        $h_{\text{cur}} \leftarrow$ ENTROPY(UNADJ)
        $S_{\text{inc}} \leftarrow \{k$: ENTROPY(UNADJ $+ (x, k)) > h_{\text{cur}}\}$
        $k_{\text{adj}} \leftarrow \operatorname*{argmax}_{k \in S_{\text{inc}}} W_k x$

    UNADJ ← UNADJ $+ (x, k_{\text{unadj}})$
    return $k_{\text{adj}}$
```

Implementation notes

As we plan to use small to medium values of K, we implement W as a dense matrix. Efficient summations and inner products with sparse vectors are straightforward. We efficiently rescale W and its columns in constant time by using the standard trick of keeping a scaling factor a along with an unscaled matrix $\tilde{W}$ such that $W = a\tilde{W}$. We compute the current assignment entropy in $O(K)$ time by storing the past positive instance assignments in a circular buffer augmented with a K length table referencing the number of assigned instances for each sub-classifier. We update this data structure in constant time by incrementing or decrementing at most two elements of the counts table every time we receive an instance. We make our C++11 implementation of the CPM, together with friendly Python bindings available to the community (Kantchelian 2014).

4.4 Non-Adversarial Evaluation

We turn to the non-adversarial evaluation of the Convex Polytope Machine. We use seven data sets all available on the LibSVM repository page (Fan 2011). Table 4.1 provides a brief description of each dataset.

Name	Features	Train Size	Test Size	Domain
MNIST-2	784	60,000	10,000	handwritten digit "2" detection
MNIST8m-2	784	8,100,000	10,000	handwritten digit "2" detection
URL	>2,300,000	1,198,065	1,198,065	malicious URL detection
RCV1-bin	47,236	20,242	677,399	text topic detection
IJCNN1	22	49,990	91,701	engine misfiring detection
A9A	123	32,561	16,281	credit scoring
KDDA	20,216,830	8,407,752	510,302	student performance prediction

Table 4.1: Summary description of our evaluation datasets.

The MNIST data set was compiled by LeCun et al. (1998) and consists of labeled handwritten digits encoded in 28×28 gray scale pictures. The MNIST8m data set consists of 8,100,000 pictures obtained by applying various random deformations to MNIST training instances MNIST (Loosli et al. 2007). Both the MNIST and MNIST8m are multi-class datasets, where the natural task is to find which of the ten digits is represented in the picture. Since our focus is on binary classification, we transform this task into a binary task by distinguishing the digit two from any other digit. To do so, we effectively relabel each instance as $y = 1$ when the true class is digit 2, and $y = -1$ otherwise. We call those transformed datasets MNIST-2 and MNIST8m-2 respectively.

The URL dataset was collected by Ma et al. (2009) in the process of creating a malicious URL detector. Each instance of this dataset is timestamped and we carve out temporally consistent training and testing sets by splitting the full dataset in two halfs such that the most recent training instance precedes the oldest testing instance.

The RCV1-bin dataset corresponds to the binary classification task of separating *corporate* and *economics* categories from *government* and *markets* categories on the RCV1 data set of news articles (Lewis et al. 2004).

The IJCNN1 dataset is set up for detecting the misfirings of a combustion engine from a collection of indirect physical measurements (Prokhorov 2001). The A9A dataset (Kohavi 1996) defines the task of predicting high annual income from census data, and finally the KDDA (Stamper et al. 2010) dataset is developed for predicting student performance on tests.

Parameter Tuning

All seven datasets have well defined training and testing subsets. To tune each algorithms meta-parameters, namely λ and h for the CPM, C and γ for RBF-SVM, and λ for AMM, we randomly select a fixed validation subset from the training set. The size of the validation subset varies between 1,000 to 10,000 depending on the size of the training set.

For the CPM, we use a double-sided CPM as described in figure 4.3, where both CPMs share the same hyper-parameters. We start by fixing a number of iterations T and a number of hyperplanes K which will result in a reasonable execution time, effectively treating these parameters as a computational budget, and we experimentally demonstrate that increasing either K or T always results in a decrease of the testing error. Once these are selected, we let $h = 0$ and select the best λ in $\{T^{-1}, 10 \times T^{-1}, \dots, 10^4 \times T^{-1}\}$. We then choose h from $\{0, \log K/10, \log 2K/10, \dots, \log 9K/10\}$, effectively performing a one-round coordinate descent on λ, h. To test the effectiveness of our empirical entropy-driven assignment procedure, we mute the mechanism by also testing with $h = 0$.

The AMM (Djuric et al. 2013) has three parameters to adjust excluding T and the equivalent of K, two of which control the weight pruning mechanism and are left set at default values. We only adjust λ. Contrary to the CPM, we do not observe the testing error of the AMM to strictly decrease with the number of iterations T. We observe erratic behavior and thus we manually select the smallest T for which the mean validation error appears to reach a minimum. For RBF-SVM, we use the LibSVM (Chang and Lin 2011) implementation and perform the usual grid search on the parameter space.

Performance

Unless stated otherwise, we used one core of an Intel Xeon E5 (3.2Ghz, 64GB RAM) for experiments. Table 4.2 presents the results of experiments and shows that the CPM achieves comparable, and at times better, classification accuracy than the RBF-SVM, while working at a relatively small and constant computational budget. For the CPM, T was up to 32 million and K ranged from 10 to 100. For AMM, T ranged from 500,000 to 36 million. Across methods, the worst execution time is for the MNIST8m-2 task, where a 512 core parallel implementation of RBF-SVM runs in two days (Zhu et al. 2009), and our sequential single-core algorithm runs in less than five minutes. The AMM has significantly larger errors and/or execution times. For small training sets such as MNIST-2 and RCV1-bin, we were not able to achieve consistent results, regardless of how we set T and λ, and we conjecture that this is a consequence of the weight pruning mechanism. The results show that our empirical entropy-driven assignment procedure for the CPM leads to better solutions for all tasks. In the RCV1-bin and MNIST-2 tasks, the improvement in accuracy from using a tuned entropy parameter is 31% and 21%, respectively.

We use the MNIST8m-2 task to the study the effects of tuning T and K on the CPM. We first choose a grid of values for T, K and for a fixed regularization factor C and $h = 0$, we train a model for each point of the parameter grid, and evaluate its performance on the testing set. Note that for C to remain constant, we adjust $\lambda = \frac{1}{CT}$. We run each experiment five times and only report the

	MNIST8m-2		URL		KDDA	
	Error	Time	Error	Time	Error	Time
CPM	0.30 ± 0.023	**4m**	**1.32 ± 0.012**	**3m**	**10.38 ± 0.027**	**6m**
CPM h=0	0.35 ± 0.034	4m	1.35 ± 0.029	3m	10.40 ± 0.021	6m
RBF-SVM	0.43*	2d**	Timed out		Timed out	
AMM	0.38 ± 0.024	1hr	2.20 ± 0.067	5m	18.25 ± 6.51	53m

* for unadjusted parameters (Zhu et al. 2009)

** running on 512 processors (Zhu et al. 2009)

(a) Large Datasets

MNIST-2		IJCNN1		A9A		RCV1-bin	
Error	Time	Error	Time	Error	Time	Error	Time
0.38 ± 0.028	**2m**	3.00 ± 0.114	2m	15.15 ± 0.062	15s	**2.82 ± 0.059**	**2m**
0.46 ± 0.026	2m	Same as CPM		Same as CPM		3.69 ± 0.156	2m
0.35	7m	**1.44**	**1s**	14.96	1m	3.7	46m
2.83 ± 1.090	1m	2.84 ± 0.312	14s	15.29 ± 0.181	**12s**	15.40 ± 6.420	1m

(b) Small Datasets

Table 4.2: Error rates and running times, including both training and testing periods, for the evaluation datasets. Means and standard deviations for five runs with random shuffling of the training set are shown.

mean accuracy. Figure 4.5 shows how this mean error rate evolves as a function of both T and K. We observe two phenomena. First, for any value $K > 1$, the error rate decreases with T. Second, for large enough values of T, the error rate decreases when K increases. These two observations validate our treatment of both K and T as budgeting parameters. The observation about K also provides empirical evidence of our conjecture that large values of K do not lead to noticeable overfitting.

Multi-class Classification

We performed a preliminary multi-class classification experiment using the MNIST/MNIST8m data sets. There are several approaches for building a multi-class classifier on top of a binary classifier (Beygelzimer et al. 2009; Beygelzimer et al. 2005; Dietterich and Bakiri 1995). We used a one-vs-one (OVO) approach where we train $\binom{10}{2} = 45$ one-vs-one classifiers and classify by a majority vote rule with random tie breaking. While this approach is not optimal, it approximates achievable performance. Table 4.3 presents the error rates and execution times of one-versus-one CPM and RBF-SVM models. While RBF-SVM dominates CPM in terms of accuracy, CPM scales

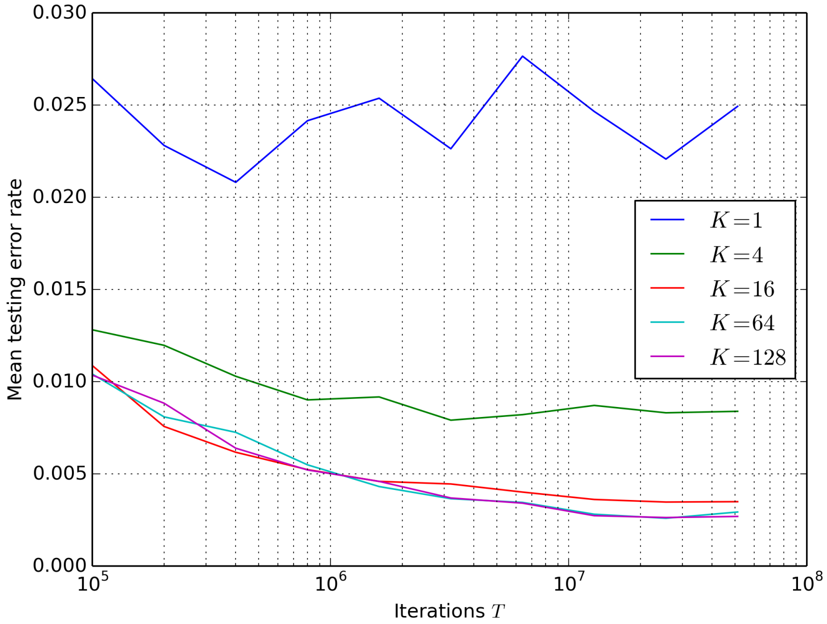


Figure 4.5: Error rate on MNIST8m-2 as a function of K, T. $C = 0.01$ and $h = 0$ are fixed.

noticeably more favorably: CPM is more than 70 times faster than RBF-SVM on the MNIST8m task, while achieving 1.03% error rate versus 0.67% for RBF-SVM.

	CPM OVO		RBF-SVM OVO	
	Error	Time	Error	Time
MNIST	1.61 ± 0.019	7min 20s	1.47	6min 43s
MNIST8m	1.03 ± 0.074	2h 35min	0.67	8 days*

* see (Loosli et al. 2007).

Table 4.3: Error rates and running times, including both training and testing periods, on the multi-class handwritten digit recognition tasks.

4.5 Exact Evasion

The non-adversarial evaluation shows that the CPM is a competitive general purpose classifier. The CPM is particularly advantageous as a drop-in replacement of linear classifiers on large-scale decision problems, as it often significantly improves classification accuracy without incurring a significant increase in computational cost.

An important question that is left unanswered by the previous evaluations is whether or not the CPM model also significantly improves evasion robustness. Because we are now working in feature space, we can practically answer this question by solving the optimal evasion problem (2.2). We present in this section an exact method for solving the constrained optimization problem (2.2) when c is a difference of two convex polytope classifiers and $\Delta(x, x') = \|x - x'\|_\rho$ for $\rho \in \{0, 1, 2, \infty\}$. In this setting, the problem is non-convex essentially because of the non-convex constraint $c(x) \neq c(x')$ and our strategy is to reduce to a Mixed Integer Linear or Quadratically Constrained Program form (MILP or MIQCP) for which efficient commercial-off-the-shelf solvers exist (Wolsey 2007). We now present the reduction of problem (2.2) to a set of mixed integer inequalities and objective function. We begin by presenting the variables of the program and then introduce the constraints and objective.

Program Variables

In what follows, the program variables are always ***bolded and italicized*** to distinguish them from program constants. Variables of the same family $\boldsymbol{u}$ are indexed using the natural subscript notation $\boldsymbol{u}_i$. Let $x \in \mathbb{R}$ be the input instance for which we are computing the optimal evasion, $W^+, W^- \in \mathbb{R}^{K \times d}$ be the weight matrices and $b^+, b^- \in \mathbb{R}^K$ the constant bias terms of each CPM.

The *instance variables* $\boldsymbol{x} \in \mathbb{R}^d$ describe the solution of the optimization problem. That is, the optimal solution x' to problem (2.2) is such that for all feature dimension j, $x'_j = \boldsymbol{x}_j$. The *choice variables* $\boldsymbol{z}^+, \boldsymbol{z}^- \in \{0; 1\}^K$ model the max operator of each convex polytope classifier. These binary variables effectively choose which sub-classifier's prediction is active for the current solution $\boldsymbol{x}$. The *classifier output variables* $\boldsymbol{f}^+$,$\boldsymbol{f}^- \in \mathbb{R}$ denote the signed margin of each convex polytope classifier. To model the distance $\Delta(x, x')$, we introduce additional *cost variables* $\boldsymbol{p}$ which number and type depend on ρ, the considered distance. The cost variables are described along with the inequalities in which they appear when presenting the modelization of the objective.

Model Consistency Constraints

We now describe in the language of linear inequalities the $\mathbb{R}^d \to \mathbb{R}$ mapping that the double-sided CPM classifier performs. That is, we link together the input variables $\boldsymbol{x}$ and output variables $\boldsymbol{f}^+$,$\boldsymbol{f}^-$ using the choice variables $\boldsymbol{z}^+, \boldsymbol{z}^-$ so as to respect the semantics of the classifier.

As can be seen in Figure 4.3, at the top level, the classifier decision is simply the difference between the outputs of the two CPMs, that is, $\boldsymbol{f}^+ - \boldsymbol{f}^-$. Without loss of generality, if our initial instance x is such that $f(x) \geq 0$, the $c(x) \neq c(x')$ is equivalent to $f(x) < 0$, that is,

$$\boldsymbol{f}^+ - \boldsymbol{f}^- \leq -\epsilon$$

for a small constant guard value $\epsilon > 0$. Because the classifier f is Lipschitzian with a modulus of continuity which depends on the parameters W and b, the specific value of ϵ is not critical. We choose $\epsilon = 10^{-3}$ in our implementation.

Without loss of generality, we focus on the positive CPM $\boldsymbol{f}^+$. By definition of the max operator, $\boldsymbol{f}^+$ must dominate all sub-classifier scores $W_k^{+\mathsf{T}}\boldsymbol{x} + b_k^+$ and be equal to one of these scores. The first condition gives the following K inequalities.

$$\forall k \in [\![1, K]\!], \quad W_k^{+\mathsf{T}}\boldsymbol{x} + b_k^+ \leq \boldsymbol{f}^+$$

For the second condition, we use the choice variables to select which sub-classifier is active. Exactly one indicator variable $\boldsymbol{z}^+$ is non-zero, which we encode as:

$$\sum_{k=1}^{K} \boldsymbol{z}_k^+ = 1$$

To use those indicator variables, we employ the so-called "big-m" method (Belotti et al. 2016). Let $m > 0$ be a large constant. The following K inequalities ensure that a single sub-classifier is selected.

$$\forall k \in [\![1, K]\!], \quad W_k^{+\mathsf{T}}\boldsymbol{x} + b_k^+ + m(1 - \boldsymbol{z}_k^+) \geq \boldsymbol{f}^+ \tag{4.11}$$

For a large-enough constant m, all inequalities with zero indicator variables z_k^+ are essentially inactive, and equality ensues at the non-zero index by virtue of the simultaneous upper and lower bounding of $\boldsymbol{f}^+$. For good solving time performance, m should be the smallest value which does not exclude valid solutions.

We use the following method to determine the smallest value for m that is also safe. In practical applications, the valid input instances $\boldsymbol{x}$ live in a strict subset of $\mathbb{R}^d$. For instance, we often have $\|\boldsymbol{x}\|_2 \leq 1$ or $\|\boldsymbol{x}\|_\infty \leq 1$. In the later case, the largest and smallest possible values for $\boldsymbol{f}^+$ are:

$$f_{\max}^+ = \max_k \left\{ \sum_{j=1}^{d} |W_{k,j}^+| + b_k^+ \right\} \quad \text{and} \quad f_{\min}^+ = \min_k \left\{ \sum_{j=1}^{d} -|W_{k,j}^+| + b_k^+ \right\}$$

A quick inspection of constraints (4.11) reveal that any value of m larger than or equal to $f_{\max}^+ - f_{\min}^+$ is a priori safe. Note that this computation can be done once and for all for a given CPM classifier.

Objective

The objective of the MIP and its associated constraints take different forms depending on the considered type of distance. We treat each four case for $\Delta(x, x') = \|x - x'\|_\rho$ in order.

For the $\rho = 0$ distance, we are minimizing the number of altered feature dimensions. In this case, we have d binary variables $\boldsymbol{p} \in \{0, 1\}^d$. By convention, $\boldsymbol{p}_j = 1$ if and only if feature dimension j was modified. Our objective is thus:

$$\min \sum_{j=1}^{d} \boldsymbol{p}_j \tag{4.12}$$

We use the big-m trick again to enforce the semantics of the cost variables. For a large-enough constant $m' > 0$, we have the following $2d$ constraints.

$$\forall j \in [\![1, d]\!], \quad x_j - \boldsymbol{x}_j \leq m'\boldsymbol{p}_j \quad \text{and} \quad \boldsymbol{x}_j - x_j \leq m'\boldsymbol{p}_j \tag{4.13}$$

The smallest safe value for m' is again dictated by our bounds in feature space. For instance, if every admissible instance x is such that $||x||_\infty \leq 1$, a safe value for m' is $1 - (-1) = 2$.

The $\rho = 1$ case is quite similar. We can continuously relax the variables $\boldsymbol{p} \in \mathbb{R}_+^d$ and fix $m' = 1$ in constraints (4.13). Each $\boldsymbol{p}_j$ measures the absolute value of the modification of feature dimension i, and the objective (4.12) remains unchanged.

For the $\rho = 2$ case, our program equivalently minimizes the square of the L_2 distance. As for the $\rho = 1$ case, the $\boldsymbol{p}$ variables are continuous and non-negative and the objective remains as before (4.12). Constraints (4.13) are replaced by d quadratic constraints, turning the MILP into a MIQCP.

$$\forall j \in [\![1, d]\!], \quad \left(x_j - \boldsymbol{x}_j\right)^2 \leq \boldsymbol{p}_j \quad \Leftrightarrow \quad \boldsymbol{x}_j^2 - 2x_j\boldsymbol{x}_j + x_j^2 \leq \boldsymbol{p}_j$$

Finally the $\rho = \infty$ case is obtained as follows. We define a single continuous non-negative variable $\boldsymbol{p}$ which represents the maximum absolute deformation over all feature dimensions. The objective of the MILP is simply $\min \boldsymbol{p}$ and there are $2d$ linear constraints as follows.

$$\forall j \in [\![1, d]\!], \quad x_j - \boldsymbol{x}_j \leq \boldsymbol{p} \quad \text{and} \quad \boldsymbol{x}_j - x_j \leq \boldsymbol{p}$$

Summary

Our MIP reduction has $\Theta(K + d)$ variables and constraints, and $\Theta(Kd)$ non-zero entries in the corresponding constraint matrix when W^+, W^- are dense. When $\rho \neq 2$, our reduction is an MILP and in the case of the Euclidean distance, the reduction is an MIQCP. Hence, we can use any modern off-the-shelf MIP solver which supports quadratic constraints to compute optimal evasion instances under problem (2.2).

We defer to Chapter 6 the experimental estimation of the evasion robustness using our reduction. This presentation choice allows us to introduce additional evasion and hardening techniques for different model classes before laying out a comprehensive evasion susceptibility comparison between classifiers.

4.6 Open Problems

Drawing on the connection between evasion robustness and large-margin learning, we propose a novel algorithm for convex polytope separation that provides larger margins than a single linear classifier, while still enjoying the computational effectiveness of a simple linear separator. Our algorithm learns a bounded number of linear classifiers simultaneously. On large data sets, the CPM outperforms RBF-SVM and AMM, both in terms of running times and error rates. Furthermore, by not pruning the number of sub-classifiers used, CPM is algorithmically simpler than AMM.

CPM avoids such complications by having little tendency to overfit the data as the number K of sub-classifiers increases, as we shown empirically. Additionally, the model class of convex polytope classifiers lends itself well to the computation of exact optimal evasions. This in turns enables the systematical study of the evasion robustness of CPM.

K and Generalization Power

Although the Rademacher-based complexity analysis suggests that the CPM overfitting error increases in $O(\sqrt{K})$, the practical evaluation shows that when using the stochastic gradient descent training method, increasing K does not lead to worse testing time accuracy. This suggests that the bounds derived in Theorem 1 are lose for the CPM. A stronger bound exhibiting a weak, if any, dependency on K might exist for the SGD-based training procedure. Alternatively, finding a probability distribution P for which the CPM overfits with the predicted $\sqrt{K}$ dependency would guarantee the tightness of the bound.

Heuristic Assignment Algorithm

Our stochastic gradient descent training procedure includes a heuristic instance to sub-classifier matching function which balances the assignment rates of the K polytope faces. While we empirically find that the resulting diversification of the sub-classifiers is often beneficial, it might be possible to ground the assignment balancing heuristic on more fundamental principles. On a related note, additional experiments on convex polytope machines that maximize the effective margin have shown a noticeably degraded accuracy compared to using the total margin. Understanding the reason for the poor performance of the effective margin may ultimately result in a novel and improved margin metric.

Chapter 5

Evasion and Hardening of Tree Ensembles

5.1 Introduction

Chapters 3 and 4 have introduced two defense techniques against evasion attacks. In Chapter 3, we proposed an evasion defense at the feature extraction level for the specific application domain of comment spam. In Chapter 4, we derived a novel model class for generic machine learning classification on large-scale datasets. The construction of the model class is grounded in an evasion robustness requirement which we cast into a large-margin approach.

In this chapter, we explore a third venue for hardening a generic machine learning algorithm against evasions: iterated re-training with artificially generated evading instances. To this end, we fix our model class to tree ensembles. Several factors explain the choice of tree ensembles.

While prior work extensively studies the evasion problem on differentiable models by means of gradient descent, those results are reported in an essentially qualitative fashion. Further, non-differentiable, non-continuous models have received very little attention. Tree ensembles as produced by boosting or bagging (Breiman 1996) is perhaps the most important model class from this family as they are often able to achieve competitive performance and enjoy good adoption rates in both industrial and academic contexts. Frustratingly, little is known about the evasion susceptibility of those models, and anecdotal claims of both robustness (Smutz and Stavrou 2012; Smutz and Stavrou 2016) and brittleness (Srndic and Laskov 2014) are simultaneously reported.

In this chapter, we present the first heuristic and exact evasion algorithms for tree ensembles. Our exact evasion method relies on a Mixed Integer Linear Program solver and enables precise quantitative robustness statements. Our approximate evasion algorithm is based on *symbolic prediction*, a fast and novel method for computing finite differences for tree ensemble models.

In addition, unlike their differentiable counterparts, tree ensembles are a natural candidate for evasion hardening by adversarial iterated re-training. Specifically, when building a tree ensemble model by a stepwise boosting algorithm, we can inexpensively modify the training set effectively used at each step by adding fresh evading instances to the original training dataset. Because those evading instances are discarded between each round, this procedure does not grows the training dataset itself, but rather grows the classifier. We call the resulting iterative model “patching” *adversarial boosting* because of its formal and operational similarity to regular boosting. The work presented in this chapter was first reported in (Kantchelian et al. 2016).

Related Work

To the best of our knowledge, there exists no prior work for the case where the adversary has full knowledge of the classifier's parameters and there is a unique paper tackling the case of unlimited query access to the model. The authors of (W. Xu et al. 2016) present a genetic algorithm for finding malicious PDF instances which evade detection. The evaluation of the fitness function is done by querying the classifier. In contrast to (W. Xu et al. 2016), our exact algorithm guarantees optimality of the solution, and our approximate algorithm performs a fast coordinate descent without the additional tuning and hyper-parameters that a genetic algorithm requires.

5.2 Tree Ensemble Models

A sum-ensemble of trees model $f : \mathbb{R}^n \to \mathbb{R}$ consists of a set $\mathcal{T}$ of regression trees. We follow the conventions set in the description of the regression tree and tree ensembles model class in Chapter 2. In this chapter, we consider the case of single-feature threshold predicates of the form $x_j < \tau$ or equivalently $x_j > \tau$, where $1 \leq j \leq d$ and $\tau \in \mathbb{R}$ are fixed model parameters. This restriction excludes oblique decision trees where predicates simultaneously involve several feature variables. We however note that oblique trees are seldom used in ensemble classifiers, partially because of their relatively high construction cost and complexity (Norouzi et al. 2015). Before describing our generic approach for solving the optimal evasion problem, we first state a simple worst-case complexity result for problem (2.2).

5.3 Theoretical Hardness of Evasion

For a given tree ensemble model f, finding an $x \in \mathbb{R}^d$ such that $f(x) > 0$ (or $f(x) < 0$ without loss of generality) is NP-complete. That is, irrespectively of the choice for Δ, the optimal evasion problem (2.2) requires solving an NP-complete feasibility subproblem.

We now give a proof of this fact by reduction from 3-SAT. First, given an instance x, computing the sign of $f(x)$ can be done in time at most proportional to the model size. Thus the feasibility problem is in NP. It is further NP-complete by a linear time reduction from 3-SAT as follows. We encode in x the assignment of values to the variables of the 3-SAT instance S. By convention, we choose $x_j > 0.5$ if and only if variable j is set to true in S. Next, we construct f by arranging each clause of S as a binary regression tree. Each regression tree has exactly one internal node per level, one for each variable appearing in the clause. Each internal node holds a predicate of the form $x_j > 0.5$ where j corresponds to a clause variable. The nodes are arranged such that there exists a unique prediction path corresponding to the falseness of the clause. For this path, the prediction value of the leaf is set to the opposite of the number of clauses in S, which is also the number of trees in the reduction. The remaining leaves predictions are set to 1. Figure 5.1 illustrates this construction on an example.

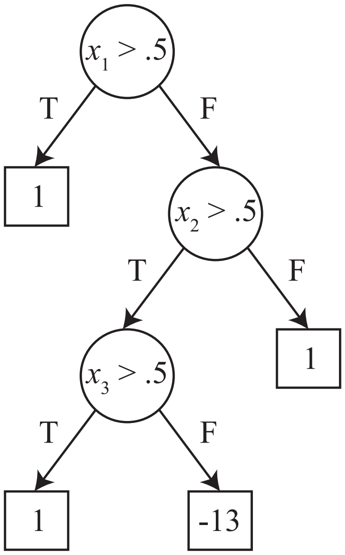


Figure 5.1: Regression tree for the clause $x_1 \vee \neg x_2 \vee x_3$. In this example, S has 13 clauses.

It is easy to see that S is satisfiable if and only if there exists x such that $f(x) > 0$. Indeed, a satisfying assignment for S corresponds to x such that $f(x) = |\mathcal{T}| > 0$ and any non-satisfying assignment for S corresponds to x such that $f(x) \leq -1 < 0$ because there is at least one false clause which corresponds to a regression tree which output is $-|\mathcal{T}|$.

While we can not expect an efficient algorithm for solving all instances of problem (2.2) unless P = NP, it may be the case that tree ensemble models as produced by common learners such as gradient boosting or random forests are practically easy to evade. We now turn to an algorithm for optimally solving the evasion problem for tree ensemble classifiers.

5.4 Exact Evasion

Let f be a sum-ensemble of trees and $x \in \mathbb{R}^d$ an initial instance. We present a reduction of problem (2.2) into a Mixed Integer Linear Program (MILP). In contrast to the CPM reduction presented in Chapter 4, this reduction avoids introducing constraints with "big-m" constants at the cost of a slightly more complex solution encoding. We experimentally find that our reduction produces tight formulations and acceptable running times for all common models f.

In what follows, we present the mixed integer program by defining three groups of MILP variables: the **predicate variables** encode the state (true or false) of all predicates, the **leaf variables** encode which prediction leaf is active in each tree, and the optional **objective variable** for the case where Δ is defined as the L_∞ norm.

We then introduce three families of constraints: the **predicates consistency constraints** enforce the logical consistency between predicates, the **leaves consistency constraints** enforce the logical consistency between prediction leaves and predicates, and the **model mislabel constraint** enforces the condition $c(x) \neq c(x')$, or equivalently that $f(x') > 0$ or $f(x') < 0$ depending on the sign of $f(x)$. Finally we reduce the objective of (2.2) by relating the predicate variables to the value of $\Delta(x, x')$ in the treatment of the **objective**.

Program Variables

For clarity, MILP variables are ***bolded and italicized*** throughout. Our reduction uses three families of variables.

- At most $\sum_{T \in \mathcal{T}} |T.\text{nodes}|$ binary variables $\boldsymbol{p}_k \in \{0; 1\}$ ($\underline{\text{p}}$redicates) encoding the state of the predicates. Our implementation sparingly create those variables: if any two or more predicates in the model are logically equivalent, their state is represented by a single variable. For example, the state of $x'_j < 0$ and $-x'_j > 0$ would be represented by the same variable.

- $\sum_{T \in \mathcal{T}} |T.\text{leaves}|$ continuous variables $0 \leq \boldsymbol{l}_k \leq 1$ ($\underline{\text{l}}$eaves) encoding which prediction leaf is active in each tree. The MILP constraints force exactly one $\boldsymbol{l}_k$ per tree to be non-zero with $\boldsymbol{l}_k = 1$. The $\boldsymbol{l}$ variables are thus implied binary in any solution but are nonetheless typed continuous to narrow down the choice of branching variable candidates during branch-and-bound, and hence improve solving time.

- At most one non-negative continuous variable $\boldsymbol{b}$ ($\underline{\text{b}}$ound) for expressing the distance $\Delta(x, x')$ of problem (2.2) when Δ is the L_∞ distance. This variable is first used in the **objective** subsection.

In what follows, we illustrate our reduction by using a model with a single regression tree as represented in figure 5.2.

Predicates consistency

Without loss of generality, each predicate variable $\boldsymbol{p}_k$ corresponds to the state of a predicate of the form $x_j < \tau_k$. If two variables $\boldsymbol{p}_k$ and $\boldsymbol{p}_{k'}$ correspond to predicates over the same variable $x_j < \tau_1$ and $x_j < \tau_2$, then $\boldsymbol{p}_k$ and $\boldsymbol{p}_{k'}$ can take inconsistent values without additional constraints. For instance, if $\tau_1 < \tau_2$, then $\boldsymbol{p}_k = 1$ and $\boldsymbol{p}_{k'} = 0$ would be logically inconsistent because $x_j < \tau_1 \Rightarrow x_j < \tau_2$, but any other valuation is possible for $\boldsymbol{p}_k$ and $\boldsymbol{p}_{k'}$.

For each feature variable x'_j, we can ensure the consistency of all $\boldsymbol{p}$ variables which reference a predicate over x'_j by adding $K - 1$ inequalities enforcing the implicit implication constraints between the predicates, where K is the number of $\boldsymbol{p}$ variables referencing x_j. For a given x'_j, let $\tau_1 < \cdots < \tau_K$ be the sorted thresholds of the predicates over x'_j. Let $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_K$ be the MILP variables corresponding to predicates $x'_j < \tau_1, \ldots, x'_j < \tau_K$. A valuation of $(\boldsymbol{p}_k)_{k=1..K}$ is consistent if and only if $\boldsymbol{p}_1 = 1 \Rightarrow \cdots \Rightarrow \boldsymbol{p}_K = 1$. Thus the consistency constraints are:

$$\boldsymbol{p}_1 \leq \boldsymbol{p}_2 \leq \cdots \leq \boldsymbol{p}_K$$

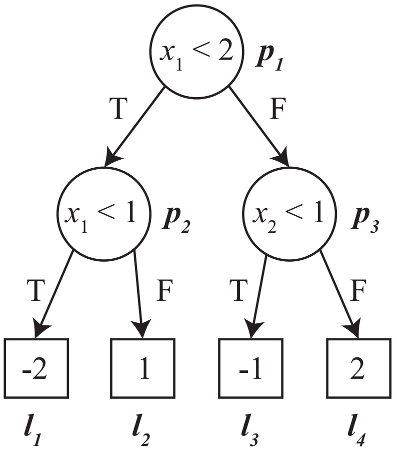


Figure 5.2: Regression tree for the reduction example. Predicate variables $\boldsymbol{p}$ and leaf variables $\boldsymbol{l}$ are shown next to their corresponding internal and leaf nodes. There are $n = 2$ continuous features. The leaf predictions are -2, 1, 1 and 2.

When the feature variables x'_j are binary-valued, there is a single $\boldsymbol{p}_k$ variable associated to a feature variable: all predicates $x'_j < \tau$ with $0 < \tau < 1$ are equivalent. Generally, tree building packages generate a threshold of 0.5 in this situation. This is however implementation dependent and we can simplify the formulation with additional knowledge of the value domain x'_j is allowed to take. In the worst case, every predicate holds over the same feature variable and no two thresholds are equal. In this case, we need to introduce exactly $\sum_{T \in \mathcal{T}} |T.\text{nodes}| - 1$ inequality constraints involving two variables.

In our toy example in figure 5.2, variables $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ refer to the same feature dimension 1 and are thus not independent. The predicate consistency constraint in this case is:

$$\boldsymbol{p}_2 \leq \boldsymbol{p}_1$$

and no other predicate consistency constraint is needed.

Leaves consistency

These constraints bind the $\boldsymbol{p}$ and $\boldsymbol{l}$ variables so that the semantics of the regression trees are preserved. Each regression tree has its own independent set of leaves consistency constraints. We construct the constraints such that the following properties hold:

(i) if $\boldsymbol{l}_k = 1$, then every other $\boldsymbol{l}_{k' \neq k}$ variable within the same tree is zero, and

(ii) if a leaf variable $\boldsymbol{l}_k$ is 1, then all predicate variables $\boldsymbol{p}$ encountered in the prediction path of the corresponding leaf are forced to be either zero or one in accordance with the semantics of the prediction path, and

(iii) exactly one $\boldsymbol{l}_k$ variable per tree is equal to 1. This property is needed because (i) does not force any $\boldsymbol{l}_k$ to be non-zero.

Enforcing property (i) is done using a classic exclusion constraint. If $\boldsymbol{l}_1, \ldots, \boldsymbol{l}_K$ are the K leaf variables for a given tree, then the following equality constraint enforces (i):

$$\sum_{k=1}^{K} \boldsymbol{l}_k = 1 \tag{5.1}$$

For our toy example, this constraint is:

$$\boldsymbol{l}_1 + \boldsymbol{l}_2 + \boldsymbol{l}_3 + \boldsymbol{l}_4 = 1$$

Enforcing property (ii) requires two constraints per internal node. Let us start at the root node ROOT. Let $\boldsymbol{p}_{\text{ROOT}}$ be the variable corresponding to the root predicate. Let $\boldsymbol{l}_1^T, \ldots, \boldsymbol{l}_{K_T}^T$ be the variables corresponding to the leaves of the subtree rooted at ROOT.TRUE, and $\boldsymbol{l}_1^F, \ldots, \boldsymbol{l}_{K_F}^F$ the variables for the subtree rooted at ROOT.FALSE. The root predicate is true if and only if the active prediction leaf belongs to the subtree rooted at ROOT.true. In terms of the MILP reduction, this means that $\boldsymbol{p}_{\text{ROOT}}$ is equal to one if and only if one of the leaf variables of the true subtree is set to one. Similarly on the false subtree, $\boldsymbol{p}_{\text{ROOT}}$ is zero if and only if one of the leaf variables of the false subtree is set to one. Because only one leaf can be non-zero, these constraints can be written as:

$$\boldsymbol{p}_{\text{ROOT}} = 1 - \sum_{k=1}^{K_F} \boldsymbol{l}_k^F \quad \text{and} \quad \boldsymbol{p}_{\text{ROOT}} = \sum_{k=1}^{K_T} \boldsymbol{l}_k^T$$

The case of internal nodes is identical, except that if and only ifs are weakened to single side implications. Indeed, unlike the root case, it is possible that no leaf in either subtree might be an active prediction leaf. For an internal node NODE, let $\boldsymbol{p}_{\text{NODE}}$ be the variable attached to the node, $\boldsymbol{l}^T$ and $\boldsymbol{l}^F$ the variables attached to leaves of the true and false subtrees rooted at NODE.TRUE and NODE.FALSE. The constraints are:

$$\boldsymbol{p}_{\text{NODE}} \leq 1 - \sum_{k=1}^{K_F} \boldsymbol{l}_k^F \quad \text{and} \quad \boldsymbol{p}_{\text{NODE}} \geq \sum_{k=1}^{K_T} \boldsymbol{l}_k^T$$

In our toy example, we have three internal nodes and thus six constraints. The constraints associated with the root, the leftmost and rightmost internal nodes are respectively:

$$\begin{aligned} \boldsymbol{p}_1 = 1 - (\boldsymbol{l}_3 + \boldsymbol{l}_4) \quad &\text{and} \quad \boldsymbol{p}_1 = \boldsymbol{l}_1 + \boldsymbol{l}_2 \\ \boldsymbol{p}_2 \leq 1 - \boldsymbol{l}_2 \quad &\text{and} \quad \boldsymbol{p}_2 \geq \boldsymbol{l}_1 \\ \boldsymbol{p}_3 \leq 1 - \boldsymbol{l}_4 \quad &\text{and} \quad \boldsymbol{p}_3 \geq \boldsymbol{l}_3 \end{aligned}$$

Finally, property (iii) automatically holds given the previously defined constraints. To see this, one can walk down the prediction path defined by the $\boldsymbol{p}$ variables and notice that at each level, the leaves values of one of the subtree rooted at the current node must be all zero. For instance, if $\boldsymbol{p}_{\text{NODE}} = 1$, then we have

$$\sum_{k=1}^{K_F} \boldsymbol{l}_k^F \leq 0 \quad \Rightarrow \quad \boldsymbol{l}_1^F = \boldsymbol{l}_2^F = \cdots = \boldsymbol{l}_{K_F}^F = 0$$

At the last internal node, exactly two leaf variables remain unconstrained, and one of them is pushed to zero. By the exclusion constraint (5.1), the remaining leaf variable must be set to 1.

Model mislabel

Without loss of generality, consider an original instance x such that $f(x) \geq 0$. In order for x' to be an evading instance, we must have $f(x') < 0$. Encoding the model output $f(x')$ is straightforward given the leaf variables $\boldsymbol{l}$. The output of each regression tree is simply the weighted sum of its leaf variables, where the weight of each variable $\boldsymbol{l}_k$ corresponds to the prediction value v_k of the associated leaf. Hence, $f(x')$ is the sum of $|\mathcal{T}|$ weighted sums over the $\boldsymbol{l}$ variables and the following constraint enforces $f(x') < 0$:

$$\sum_k v_k \boldsymbol{l}_k \leq -\epsilon$$

where $\epsilon > 0$ is a small constant which acts as a guard value.

For our running example, the mislabeling constraint is:

$$-2\boldsymbol{l}_1 + \boldsymbol{l}_2 - \boldsymbol{l}_3 + 2\boldsymbol{l}_4 \leq -\epsilon$$

Objective

Finally, we need to translate the objective $\Delta(x, x')$ of problem (2.2). We rely on the predicate variables $\boldsymbol{p}$ in doing so. For any distance L_ρ with $\rho \in \mathbb{N}$, there exists weights w_k and a constant C such that the MILP objective can be written as:

$$\sum_k w_k \boldsymbol{p}_k + C$$

We now describe the construction of $(w_k)_k$ and C. Recall that for each feature dimension $1 \leq j \leq d$, we have a collection of predicate variables $(\boldsymbol{p}_k)_{k=1..K}$ associated with predicates $x'_j < \tau_1, \ldots, x'_j < \tau_K$ where the thresholds are sorted $\tau_1 < \cdots < \tau_K$. Thus, the $\boldsymbol{p}$ variables effectively encode the interval to which x'_j belongs to, and any feature value within the interval will lead to the same prediction $f(x')$. There are exactly $K + 1$ distinct possible valuations for the binary variables $\boldsymbol{p}_1 \leq \boldsymbol{p}_2 \leq \cdots \leq \boldsymbol{p}_K$ and the value domain mapping $\phi : \boldsymbol{p} \to (\mathbb{R} \cup \{-\infty; \infty\})^2$ is:

$$\begin{aligned} &x'_j \in \phi(\boldsymbol{p}) = [\tau_k, \tau_{k+1}) \\ &\quad \text{where } k = \max\,\{k' \in [\![0, K+1]\!] \mid \boldsymbol{p}_{k'} = 0\} \end{aligned}$$

where by convention $\boldsymbol{p}_0 = 0, \boldsymbol{p}_{K+1} = 1$ and $\tau_0 = -\infty$, $\tau_{K+1} = \infty$. Setting aside the L_∞ case for now, consider $\rho \in \mathbb{N}$ the norm we are interested in for Δ. Instead of directly minimizing $\|x - x'\|_\rho$, our formulation equivalently minimizes $\|x - x'\|_\rho^\rho$. By minimizing the latter, we are able to consider the contributions of each feature dimension independently:

$$\|x - x'\|_\rho^\rho = \sum_{j=1}^{d} |x_j - x'_j|^\rho$$

We take $0^0 = 0$ by convention. At the optimal solution, $|x_j - x'_j|^\rho$ can only take $K+1$ distinct values. Indeed, if x'_j and x_j belong to the same interval, then $x'_j = x_j$ minimizes the distance along feature j, and this distance is zero. If x'_j and x_j do not belong to same interval, then setting x'_j at the bound of $\phi(\boldsymbol{p})$ that is closest to x_j minimizes the distance along j. If $\phi(\boldsymbol{p}) = [\tau_k, \tau_{k+1})$, this distance is simply equal to $\min\{|x_j - \tau_k|^\rho, |x_j - \tau_{k+1}|^\rho\}$. Note that because of the right-open interval, the minimum distance is actually an infimum. In our implementation, we simply use a guard value $\epsilon = 10^{-4}$ of the same magnitude order than the numerical tolerance of the MILP solver.

Hence, we can express the minimization objective of problem (2.2) as a weighted sum of $\boldsymbol{p}$ variables without loss of generality. Let $0 \leq k \leq K+1$ be the indices such that $x_j \in [\tau_k, \tau_{k+1})$. Let $(w_k)_{k=0..K+1}$ such that for any valid valuation of $\boldsymbol{p}$ we have $\sum_{k=0}^{K+1} w_k \boldsymbol{p}_k = \inf_{\alpha \in \phi(\boldsymbol{p})} |x_j - \alpha|^\rho$. By the discussion above and exhaustively enumerating the $K+1$ valuations of $\boldsymbol{p}$, w is the solution to the following $K+1$ equations:

$$\begin{aligned}
w_{K+1} &= |x_j - \tau_K|^\rho \\
w_K + w_{K+1} &= |x_j - \tau_{K-1}|^\rho \\
&\dots \\
w_{k+1} + \dots + w_{K+1} &= |x_j - \tau_{k+1}|^\rho \\
w_k + w_{k+1} + \dots + w_{K+1} &= 0 \\
w_{k-1} + w_k + w_{k+1} + \dots + w_{K+1} &= |x_j - \tau_k - \epsilon|^\rho \\
&\dots \\
w_1 + w_2 + w_3 + \dots + w_{K+1} &= |x_j - \tau_2 - \epsilon|^\rho \\
w_0 + w_1 + w_2 + w_3 + \dots + w_{K+1} &= |x_j - \tau_1 - \epsilon|^\rho
\end{aligned}$$

Note that this system of linear equations is already in triangular form and obtaining the w values is immediate. To obtain the full MILP objective, we repeat this process for every feature $j \in [\![1, d]\!]$ and take the sum of all weighted sums of subsets of $\boldsymbol{p}$.

Finally, for the L_∞ case, we use one continuous variable $\boldsymbol{b}$. We introduce d additional constraints to the formulation, one for each feature dimension j. As per the previous discussion, we can generate the weights w as in the $\rho = 1$ case:

$$\sum_{k=0}^{K+1} w_k \boldsymbol{p}_k = \inf_{\alpha \in \phi(\boldsymbol{p})} |x_j - \alpha|$$

The additional constraint on dimension j is then:

$$\sum_{k=0}^{K+1} w_k \boldsymbol{p}_k \leq \boldsymbol{b}$$

and the MILP objective simply becomes the variable $\boldsymbol{b}$ itself.

For our toy example, consider $(x_1 = 0, x_2 = 3)$. In the case of the L_0 distance, we have the following objective:

$$1 - \boldsymbol{p}_2 + \boldsymbol{p}_3$$

For the (squared) L_2 distance instead, the objective is, when setting the guard $\epsilon = 0$:

$$4 - 3\boldsymbol{p}_1 - \boldsymbol{p}_2 + 4\boldsymbol{p}_3$$

For the L_∞ case, our objective reduces to the variable $\boldsymbol{b}$ and we introduce d additional bounding constraints of the form $\cdots \leq \boldsymbol{b}$ where the left hand side measures $|x_j - x'_j|$ using the same technique as the $\rho = 1$ case.

Hence, the full MILP reduction of the optimal L_0-evasion for our toy instance is:

$$\begin{aligned}
\min_{\boldsymbol{p},\boldsymbol{l}}\ & 1 - \boldsymbol{p}_1 + \boldsymbol{p}_2 & \\
\text{s.t.}\ & \boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3 \in \{0; 1\};\ \boldsymbol{l}_1, \boldsymbol{l}_2, \boldsymbol{l}_3, \boldsymbol{l}_4 \in [0, 1] & \\
& \boldsymbol{p}_1 \leq \boldsymbol{p}_0 & \text{predicates consistency} \\
& \boldsymbol{l}_1 + \boldsymbol{l}_2 + \boldsymbol{l}_3 + \boldsymbol{l}_4 = 1 & \text{leaves consistency} \\
& \boldsymbol{l}_1 + \boldsymbol{l}_2 = \boldsymbol{p}_0 = 1 - (\boldsymbol{l}_3 + \boldsymbol{l}_4) & \text{leaves consistency} \\
& \boldsymbol{l}_1 \leq \boldsymbol{p}_1 \leq 1 - \boldsymbol{l}_2 & \text{leaves consistency} \\
& \boldsymbol{l}_3 \leq \boldsymbol{p}_2 \leq 1 - \boldsymbol{l}_4 & \text{leaves consistency} \\
& -2\boldsymbol{l}_1 + \boldsymbol{l}_2 - \boldsymbol{l}_3 + 2\boldsymbol{l}_4 \geq 0 & \text{model mislabel}
\end{aligned}$$

Additional Constraints

Reducing problem (2.2) to a MILP allows expressing potentially complex inter-feature dependencies created by the feature extraction step ψ. For instance, consider the common case of K mutually exclusive binary features $x'_{j_1}, \ldots, x'_{j_K}$ such that in any well-formed instance, exactly one feature is non-zero. Letting $\boldsymbol{p}_k$ be the predicate variable associated with $x'_{j_k} < 0.5$, mutual exclusivity can be enforced by:

$$\sum_{k=1}^{K} \boldsymbol{p}_k = K - 1$$

5.5 Approximate Evasion

While the above reduction of problem (2.2) to an MILP is linear in the size of the model f, the actual solving time can be very significant for difficult models. Thus, as a complement to the exact method, we develop an approximate evasion algorithm to generate good quality evading instances. For this part, we exclusively focus on minimizing the L_0 distance. Our approximate evasion algorithm is based on the iterative coordinate descent procedure described in algorithm 3.

Algorithm 3 Coordinate Descent for Problem (2.2). $u^{(j)}$ is the j-th canonical basis vector of $\mathbb{R}^d$. That is, $u^{(j)}$ is a d-dimensional vector such that $u^{(j)}_j = 1$ and every other coordinate is zero.

```
Input: model $f$, initial instance $x$ (assumes $f(x) \geq 0$)

function coordinateDescent($f, x$)
    Initialize $S \leftarrow \{\}$
    while $f(x) \geq 0$ do
        $(j, \alpha) \leftarrow \underset{j' \in [\![1,d]\!],\ \alpha' \in \mathbb{R}}{\operatorname{argmin}} f(x + \alpha' u^{(j')})$
        if $f(x + \alpha u^{(j)}) \geq f(x)$ and $j \in S$ then
            break
        $x_j \leftarrow x_j + \alpha$
        $S \leftarrow S \cup \{j\}$
    return $x$
```

This algorithm greedily modifies the single best feature at each iteration until either the sign of $f(x')$ changes, or there is no new feature to modify in x. The last condition ensures that the algorithm terminates in less than d steps. Even though the resulting x is not guaranteed to be evading, we find that in practice Algorithm 3 almost always returns an evading instance.

We now present an efficient algorithm for solving the inner optimization subproblem

$$\min_{\tilde{x} : \|x - \tilde{x}\|_0 = 1} f(\tilde{x}) \tag{5.2}$$

The time complexity of a careful brute force approach is high. For balanced regression trees, the prediction time for a given instance is $O\left(\sum_{T \in \mathcal{T}} \log |T.\textsc{nodes}|\right)$. Further, for each dimension $1 \leq j \leq d$, we must compute all possible values of $f(\tilde{x})$ where $\tilde{x}$ and x only differ along dimension j. Note that because the model predicates effectively discretize the feature space, $f(\tilde{x})$ takes a finite number of distinct values. This number is no more than one plus the total number of predicates holding over feature j. Hence, we must compute $f(\tilde{x})$ for a total of $\sum_{T \in \mathcal{T}} |T.\textsc{nodes}|$ candidates $\tilde{x}$ and the total running time is $O\left(\sum_{T \in \mathcal{T}} |T.\textsc{nodes}| \times \sum_{T \in \mathcal{T}} \log |T.\textsc{nodes}|\right)$. If we denote by $|f|$ the size of the model which is proportional to the total number of predicates, the running time is $O\left(|f||\mathcal{T}| \log \frac{|f|}{|\mathcal{T}|}\right)$. Tree ensembles often have thousands of trees, making the $|f||\mathcal{T}|$ dependency prohibitively expensive.

We can efficiently solve problem (5.2) by using a dynamic programming approach. The main idea is to visit each internal node no more than once by computing what value of $\tilde{x}$ can land us at each node. We call this approach *symbolic prediction* in reference to symbolic program execution (King 1976), because we essentially move a symbolic instance $\tilde{x}$ down the regression tree and keep track of the constraints imposed on $\tilde{x}$ by all encountered predicates. Because we are only interested in $\tilde{x}$ instances that are at most one feature away from x, we can stop the tree exploration early if the current constraints imply that at least two dimensions need to be modified or more trivially, if there is no instance $\tilde{x}$ that can simultaneously satisfy all the constraints. When reaching a leaf, we report the leaf prediction value $f(\tilde{x})$ along with the pair of perturbed dimension number j and value interval for $\tilde{x}_j$ which would reach the given leaf.

To simplify the presentation of the algorithm, we introduce a SymbolicInstance data structure which keeps track of the constraints on $\tilde{x}$. This structure is initialized by x and has four methods.

- For a predicate p, .isFeasible(p) returns true if and only if there exists an instance $\tilde{x}$ such that $\|\tilde{x} - x\|_0 \leq 1$ and all constraints including p hold.
- .update(p) updates the set of constraints on $\tilde{x}$ by adding predicate p.
- .isChanged() returns true if and only if the current set of constraints imply $x \neq \tilde{x}$.
- .getPerturbation() returns the index j such that $x_j \neq x'_j$ and the admissible interval of values for $\tilde{x}_j$

It is possible to implement SymbolicInstance such that each method executes in constant time.

Algorithm 4 presents the symbolic prediction algorithm recursively for a given tree. It updates a list of elements by appending tuples to it. The first element of a tuple is the feature index j where $\tilde{x}_j \neq x_j$, the second element is the allowed right-open interval for $\tilde{x}_j$, and the last element is the prediction score $f(\tilde{x})$.

This algorithm visits each node at most once and performs at most one copy of the SymbolicInstance s per visit. The copy operation is proportional to the number of constraints in s. For a balanced tree T, the copy cost is $O(\log |T.\text{nodes}|)$, so that the total running time is $O(|T.\text{nodes}| \log |T.\text{nodes}|)$.

For each tree of the model, once the list of (dimension, interval, prediction) tuples is obtained, we substract the leaf prediction value for x from all predictions in order to obtain a score variation between $\tilde{x}$ and x instead of the score for $\tilde{x}$. With the help of an additional data structure, we can use the (dimension, interval, variation) tuples across all trees to find the dimension j and interval for $\tilde{x}_j$ which corresponds to the smallest variation $f(\tilde{x}) - f(x)$. This final search can be done in $O(L \log L)$, where L is the total number of tuples, and is no larger than $\sum_{T \in \mathcal{T}} |T.\text{leaves}|$ by construction. To summarize, the time complexity of our method for solving problem (5.2) is $O(|f| \log |f|)$, an exponential improvement over the brute force method.

5.6 Adversarial Boosting

We now turn to the presentation of our iterated retraining method for hardening tree ensembles against evasion attacks. Iterative retraining mimics the arms race between the system defender and an evading adversary. In its basic form, iterative retraining consists in the following iterative process:

Algorithm 4 Recursive definition of the symbolic prediction algorithm. For the first call, NODE is the tree root, s is a fresh SymbolicInstance object initialized on x with no additional constraints and l is an empty list.

```
Input: node NODE (either internal or leaf)
Input: $s$ of type SymbolicInstance
Input/Output: list of tuples $l$ (see description)

if NODE is a leaf then
    if $s$.isChanged() then
        $l \leftarrow l \cup \{s$.getPerturbation(), NODE.prediction$\}$
else
    if $s$.isFeasible(NODE.PREDICATE) then
        $s_T \leftarrow$ COPY($s$)
        $s_T$.UPDATE(NODE.PREDICATE)
        symbolicPrediction(NODE.TRUE, $s_T$, $l$)
    if $s$.isFeasible($\neg$NODE.PREDICATE) then
        $s$.UPDATE($\neg$NODE.PREDICATE)
        symbolicPrediction(NODE.FALSE, $s$, $l$)
```

1. build a classifier from the training set,
2. compute evasion instances with starting points in the original training set for the classifier,
3. augment the training set by adding the evasion instances,
4. repeat from step 1.

The major issue of the naive iterative retraining process is its non-scalability with respect to the size n of the original training set. Indeed, notice that the size of the effective training set grows linearly at each iteration. Considering the optimistic case of a training algorithm with linear time complexity over the size of the dataset, the total training time for K iterations is quadratic as $\Theta(nK^2)$. In practice, we find that iterative retraining requires relatively large values of K to be effective. For instance, we demonstrate results for $K \approx 10^4$ retraining rounds in Chapter 6. Thus, naive iterative retraining is only practical for the smallest training datasets sizes $n \leq 10^3$.

Adversarial boosting sidesteps the issue of the unbounded growth of the effective training set by iteratively augmenting the classifier instead of the dataset. The intuition of adversarial boosting is twofold. First, the procedure is computationally wasteful. At every round of naive iterated retraining, we throw away the last learned classifier and learn a new one from scratch. Second, the procedure is also redundant: the classifier learned from the dataset at iteration t by definition contains a description of the dataset, so that materializing both the classifier and the augmented dataset at every step is redundant.

Figure 5.3 presents the adversarial boosting framework in the context of tree ensembles. At every iteration, adversarial boosting grows the model by adding a new base classifier to the ensemble.

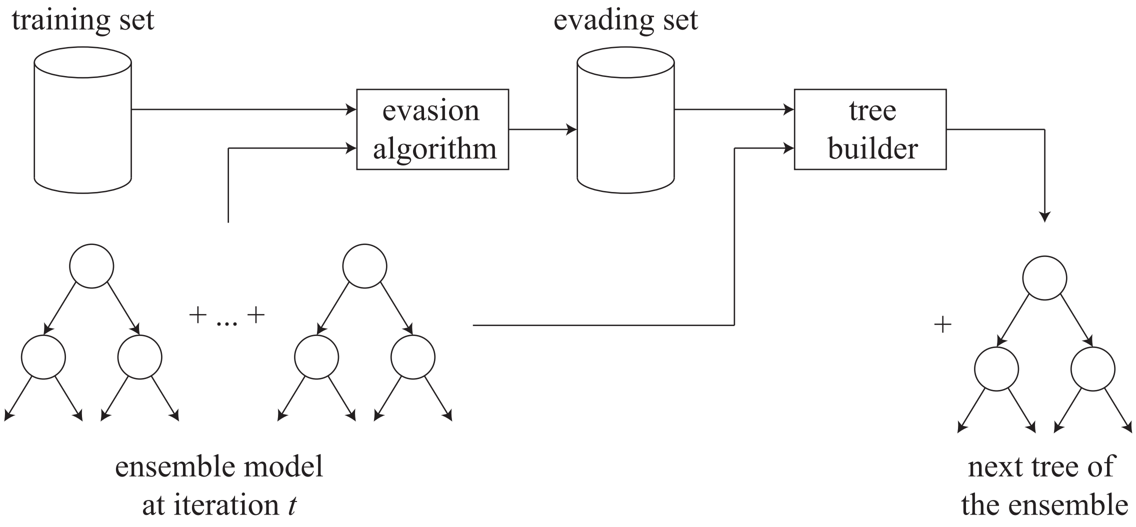


Figure 5.3: The adversarial boosting framework. We iteratively build a hardened tree ensemble by adding at every round a tree which minimizes the error of the resulting model on an evading set of instances instead of the original training data. The evading set is a pessimal version of the original training set with respect to the current ensemble model.

Unlike regular boosting in which the new base classifier is chosen to minimize the error of the resulting ensemble on the training data, adversarial boosting instead minimizes the error over a pessimal set of instances. The pessimal set of instances is constructed by computing budgeted evasions of the original training set with respect to the current model. In other words, the generated pessimal instances are hard-to-classify variations of the original training instances for the current model. Hence, adversarial boosting can be understood as iteratively finding the defects, or blind spots, of a classifier and "patching" those by adding base classifiers.

Adversarial boosting needs two components: a model builder and an evasion algorithm. The model builder can be any standard machine learning algorithm. Specifically, for a given, fixed classifier f, e.g., a tree ensemble, and for a base model class $\mathcal{F}$, e.g., regression trees, a model builder is an algorithm that finds a classifier $h \in \mathcal{F}$ that minimizes the loss of the ensembled classifier $f + h$ over the dataset. Formally, the model builder solves the following optimization problem.

$$\min_{h \in \mathcal{F}} R_{\mathcal{D}}(f + h) = \min_{h \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell \left(y_i f(x^{(i)}) + y_i h(x^{(i)}) \right)$$

In the case when $\mathcal{F}$ is the class of regression trees, Li (2012) presents an efficient and generic model builder based on the second-order Taylor development of the loss term.

The evasion algorithm finds *budgeted* adversarial deformations of the original training instances such that the altered instances are confusing for a given classifier f. In this thesis, the evasion algorithm is given a deformation bound $B > 0$ and solves the following problem for each training instance x with label y.

$$\min_{x' \,:\, \Delta(x,x') \leq B} y f(x') \tag{5.3}$$

We can readily transform both our exact and heuristic solutions to the optimal evasion problem (2.2) to instead solve the budgeted evasion problem (5.3). Because of the efficiency of modern tree builders and our heuristic L_0 evasion technique, tree ensembles are a natural candidate for hardening under the adversarial boosting framework.

5.7 Open Problems

In this chapter, we develop two algorithms for solving the optimal evasion problem (2.2) when the classifier is a sum ensemble of trees. Our first algorithm computes an exact solution for any distance metric L_ρ with $\rho \in \mathbb{N} \cup \{\infty\}$ by reduction to a mixed integer linear program. Our second algorithm finds a heuristic solution for the L_0 case by coordinate descent procedure. We develop the *symbolic prediction* algorithm, an efficient dynamic programming technique for computing all finite differences for a tree ensemble classifier.

Finally, borrowing from the concept of boosting, we present *adversarial boosting*, a general machine learning framework for hardening classifiers by iterative re-training. When used in tandem with symbolic prediction and efficient tree building techniques, adversarial boosting is a computationally practical method for hardening tree ensembles.

Extended Heuristic Evasion Algorithm

Our heuristic evasion algorithm only handles the L_0 case. Our formulation of the symbolic prediction method points to a possible extension to the general L_ρ case. For instance, instead of collecting all possible finite differences $f(x') - f(x)$ for x' that are one feature away from x, we could collect the finite differences for all x' less than a given distance θ: $\|x' - x\|_\rho \leq \theta$. In this generalized setting, we would also need to modify the coordinate descent in Algorithm 3 to heuristically choose the best perturbation among all candidate perturbations returned by symbolic prediction. Some of the relevant factors for a good choice are a small modification $\|x' - x\|_\rho$ size and a large negative finite difference $f(x') - f(x)$.

Theoretical Properties of Adversarial Boosting

Our introduction of adversarial boosting mimics the iterated evasion/retraining process that machine learning practitioners experience in practical adversarial settings. One important question for adversarial boosting, and for adversarial retraining in general is to understand the conditions under which iterated retraining succeeds at constructing both an accurate and hard to evade classifier. We suspect that those conditions involve at the minimum some characterisation of the dataset in terms of the separating distance between the two classes and the maximal admissible adversarial deformation B that the adversary can introduce.

Chapter 6

Empirical Adversarial Evaluation

6.1 Introduction

We now turn to the empirical measurement of evasion susceptibility for the classifiers considered in Chapters 4 and 5 of this thesis. We present the first adversarial evaluations for the Convex Polytope Machine developed in Chapter 4 as well as for random forests, boosted trees and adversarially boosted trees from Chapter 5. For comparison purposes, we carry the same evasion experiments for a large variety of common off-the-shelf machine learning models.

We start by describing our choice of benchmark classification task and comparative model classes. We proceed to provide both qualitative and quantitative evasion susceptibility results for all L_ρ distances and all benchmarked models. We pay special attention to tree ensembles and to the application of our exact and approximate evasion algorithms.

6.2 Experimental Setup

We choose digit recognition over the MNIST (LeCun et al. 1998) dataset as our benchmark classification task. There are three benefits in choosing MNIST as our benchmark task. First, MNIST is a well studied and recognized dataset in the machine learning community. In particular, we can be confident that MNIST is largely exempt from labeling errors. Additionally, because many classifiers perform well on MNIST, we can disentangle the problem of accurate classification from evasion-resistant classification by making sure that all considered models enjoy useful testing accuracies.

Second, the feature extractor ψ is essentially the identity function in this task: there is a trivial one-to-one mapping between image pixels and feature dimensions. This implies there are no hidden dependencies between features so that the coordinates of x can vary independently from each other. In the same vein, we can pictorially represent evading instances, and this helps understanding the models' robustness or lack of.

Third, measuring the semantic distortion $\Delta(x, x')$ between two pictures x, x' by the L_ρ distances makes intuitive sense on this task. For example, when Δ is the L_0 distance, the optimal evasion problem (2.2) measures the minimal number of pixels to modify before changing the output label of the classifier. In general, our perception of a picture remains unchanged for small enough values of the L_ρ distance, regardless of $\rho \in \mathbb{N}$.

Method

Originally, MNIST defines a multi-class classification problem over the 10 handwritten digits. We carve out a binary classification problem by focusing on distinguishing between handwritten digits “2” and “6”. Our training and testing sets respectively include 11,876 and 1,990 images and each image has $d = 28 \times 28 = 784$ gray scale pixels. The intensity of each pixel is represented by a continuous value between zero and one, so that our feature space is effectively $[0, 1]^{784}$.

As our main goal is not to compare model accuracies, but rather to obtain the best possible model for each model class, we tune the hyper-parameters so as to minimize the error on the testing set directly. In addition to the training and testing sets, we create an evaluation dataset of a hundred instances from the testing set such that every instance is correctly classified by *all* of the benchmarked models. These correctly classified instances are to serve the purpose of x, the starting point instances in the evasion problem (2.2).

Considered Models

Table 6.1 summarizes the seven benchmarked models with their salient hyper-parameters and error rates on the testing set.

Model	Parameters	Test Error	Evasion Method
Lin. L_1	$C = 0.5$	1.5%	Exact
Lin. L_2	$C = 0.2$	1.5%	Exact
BDT	1,000 trees, depth 4, $\eta = 0.02$	0.25%	Exact (Chapter 5)
RF	80 trees, max. depth 22	0.20%	Exact (Chapter 5)
CPM	double sided, $K = 30$, $C = 0.01$	0.20%	Exact (Chapter 4)
NN	60-60-30 sigmoidal (tanh) units	0.25%	Heuristic (Projected Gradient Descent)
RBF-SVM	$\gamma = 0.04$, $C = 1$	0.25%	Heuristic (Projected Gradient Descent)
BDT-R	1,000 trees, depth 6, $\eta = 0.01$	0.20%	Heuristic+Exact (Chapter 5)

Table 6.1: The benchmarked models. BDT-R is the hardened boosted trees model introduced in Chapter 5 and is further discussed in Section 6.4.

For our tree ensembles, BDT is a (gradient) boosted decision trees model in the modern XGBoost implementation (T. Chen and He 2014; T. Chen and Guestrin 2016) and RF is a random forest trained using scikit-learn (Buitinck et al. 2013). We also include the following models for comparison purposes. Lin. L_1 and Lin. L_2 are respectively a L_1 and L_2-regularized logistic regression using the LibLinear (Fan et al. 2008) implementation. RBF-SVM is a Gaussian kernel SVM trained using LibSVM (Chang and Lin 2011). NN is a three hidden layer neural network with a top logistic regression layer implemented using Theano (Bergstra et al. 2010). We use no pre-training nor drop-out for training model NN. Finally, our last benchmark model is the equivalent of a shallow neural network made of two max-out units (one unit for each class) each made of thirty linear classifiers. This model corresponds to the difference of two Convex Polytope Machines. Finally, we use Gurobi (Gurobi Optimization 2016) as the MIP solver.

Except for the two linear classifiers, all considered models have a comparable, very low error rate on this benchmark.

6.3 Evasion Susceptibility

For each learned model, and for all of the 100 correctly classified evaluation instance, we compute the optimal (or best effort) solution to the optimal evasion problem under all deformation metrics. The optimal evasions for all distances and all models but NN and RBF-SVM are optimally computed by the MIP solver with the techniques we introduce in Chapters 4 and 5. We use a classic projected gradient descent method for solving the L_1, L_2 and L_∞ evasions of NN and RBF-SVM, and address the L_0-evasion case by an iterative coordinate descent algorithm and a brute force grid search at each iteration.

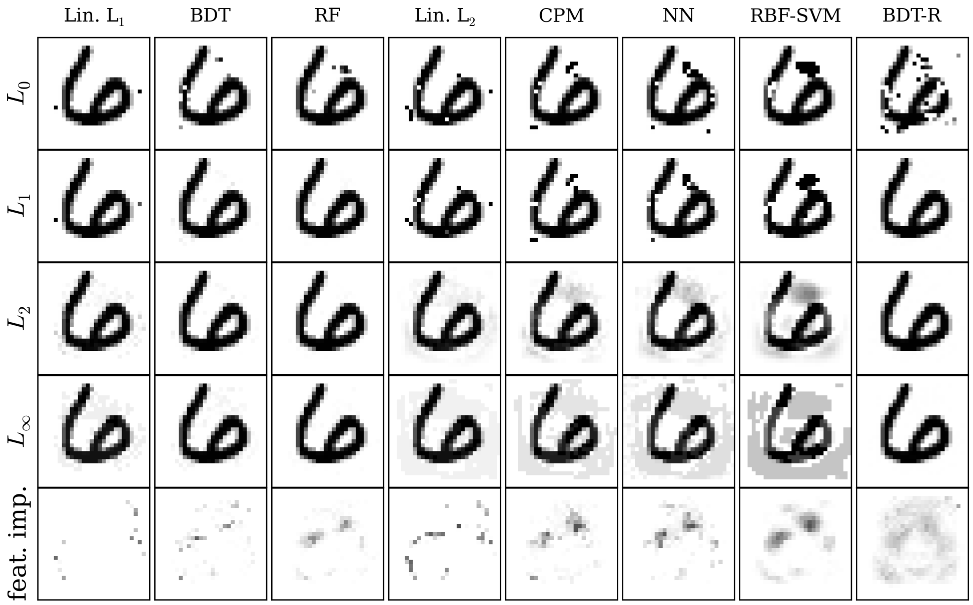


Figure 6.1: First four rows: examples of optimal or best effort evading "6" instances. Every picture is misclassified as "2" by its column model. Last row: feature importance computed as frequency of pixel modification in the L_0-evasions over the 100 evaluation instances (darker means feature is more often picked).

Figure 6.1 provides a qualitative understanding of the optimal evading instances x' across models and distance types. In this example, the initial instance x is correctly recognized as the digit “6” by all classifiers. For most models and distances, the difference between x and the closest image recognized as a “2” is insignificant.

Figure 6.2 quantitatively presents the adversarial bounds as one boxplot for each combination of model and distance. Although the tree ensembles BDT and RF have very competitive accuracies, they systematically rank at the bottom for robustness across all metrics. Remarkably, negligible L_1 or L_2 perturbations suffice to evade those models. On the other end, RBF-SVM is apparently the hardest model to evade, agreeing with the observations of (Goodfellow et al. 2015). NN and CPM exhibit very similar performance despite having quite different architectures. Unfortunately, in spite of its large-margin groundings, the CPM does not show a significant increase in evasion robustness. Finally, the L_1-regularized linear model exhibits significantly more brittleness than its L_2 counterpart. This phenomenon is explained by large weights concentrating in specific dimensions as a result of sparsity. Thus, small modifications in the heavily weighted model dimensions can result in large classifier output variations.

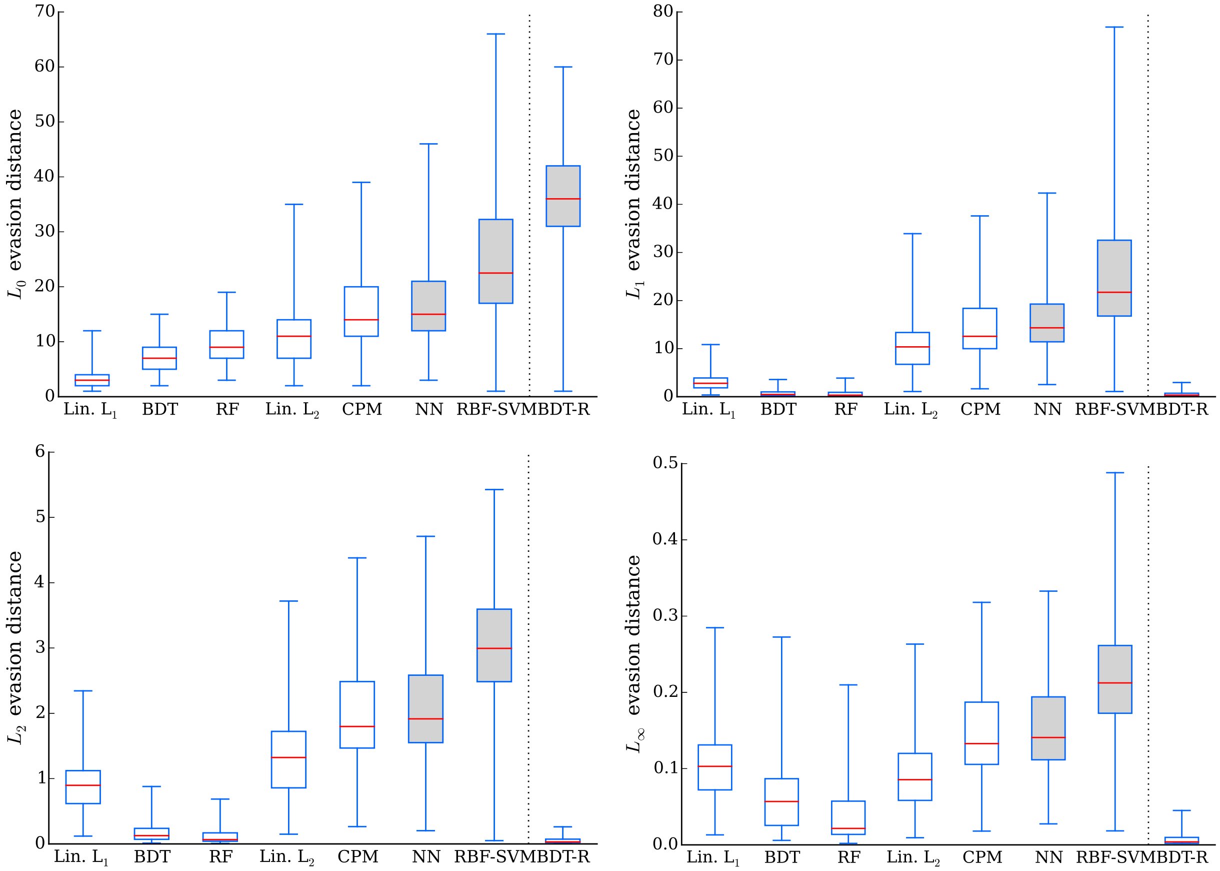


Figure 6.2: Optimal (white boxes) or best-effort (gray boxes) evasion bounds for different metrics on the evaluation dataset. The smallest bounds, 25-50% and 50-75% quartiles and largest bounds are shown. The red line is the median score. Larger scores mean more deformations are necessary to change the model prediction.

6.4 Hardening Tree Ensembles with Adversarial Boosting

We empirically demonstrate how to significantly improve the robustness of the BDT model by adding evading instances to the training set during the boosting process. At each boosting round, we use our fast *symbolic prediction*-based algorithm to create budgeted "adversarial" instances with respect to the current model and for all the 11,876 original training instances. For a given training instance x with label y and a modification budget $B \geq 1$, a budgeted "adversarial" training instance x^* is such that $\|x - x^*\|_0 \leq B$ and the margin $yf(x^*)$ is as small as possible. Here, we use $B = 28$, the size of the diagonal of the picture, as our budget. The reason is that modifying 28 pixels over 784 is not enough to morph a handwritten "2" into "6" or vice-versa. The training dataset for the current round is then formed by appending to the original training dataset these evading instances along with their correct labels, thus increasing the size of the training set by a factor two. Finally, gradient boosting produces the next regression tree which by definition minimizes the error of the augmented ensemble model on the adversarially-enriched training set. After 1,000 adversarial boosting round, our model has encountered more than eleven million adversarial instances, without ever training on more than 24,000 instances at each round.

We found that we needed to increase the maximum tree depth from four to six in order to obtain an acceptable error rate. After 1,000 iterations, the resulting model BDT-R has a slightly higher testing accuracy than BDT but still makes a few errors over the training set as can be seen in Figure 6.3. In contrast, regular boosting converges faster to a perfect classifier for the training set.

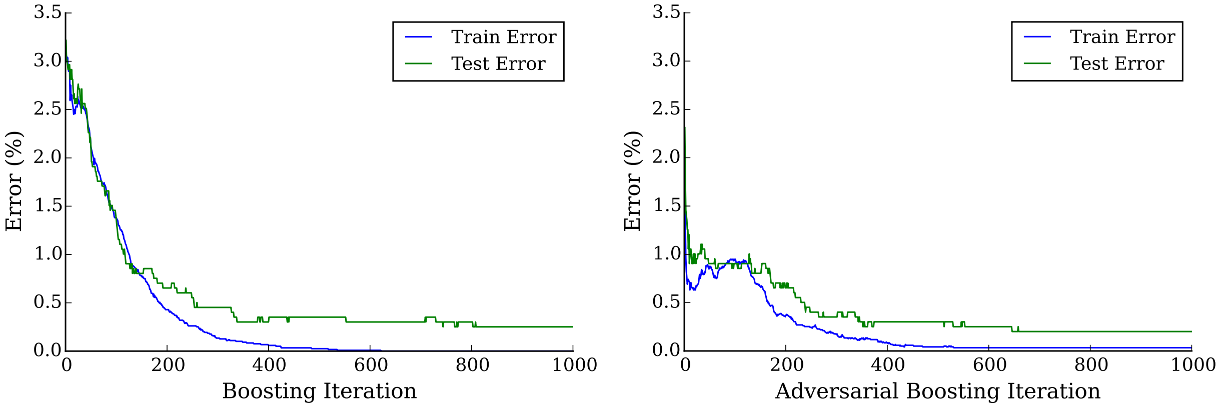


Figure 6.3: Regular and adversarial boosting errors at each iteration. Regular boosting exhibits a predictable error rate decay. In contrast, adversarial boosting starts with a significant drop in the error rate followed by a slower and more chaotic decay.

Unlike BDT, BDT-R is also extremely challenging to optimally evade using the MILP solver: the branch-and-bound search continues to expand nodes after one day on a six core Xeon 3.2GHz machine. To obtain the tightest possible evasion bound, we warm-start the solver with the solution found by the fast evasion technique and report the best solution found by the solver after an hour. Figure 6.2 shows that BDT-R is significantly more robust against L_0 evasions than our previous champion RBF-SVM and the top right image in Figure 6.1 illustrates a best-effort evasion for

BDT-R. To see if the gained robustness against L_0-evasions translates into robustness against other distances, we also measured the L_1, L_2 and L_∞ robustness bounds. Unfortunately, we found significantly lower scores on all three metrics compared to the original BDT model: hardening against L_0-evasions made the model extremely susceptible to all other types of evasions.

6.5 Quality of the Approximate Evasion Method

We also evaluate the quality of the solutions found by our heuristic L_0-evasion procedure. Figure 6.4 shows the evasion bounds found by the optimal evasion technique using the MILP solver, and our heuristic coordinate descent. Our evaluation suggests that for most of the evaluation instances, the heuristic solution is close, if not identical, to the optimal solution. It should however be noted that in at least one instance for the random forest model, the heuristic method significantly over-estimated the minimal bound.

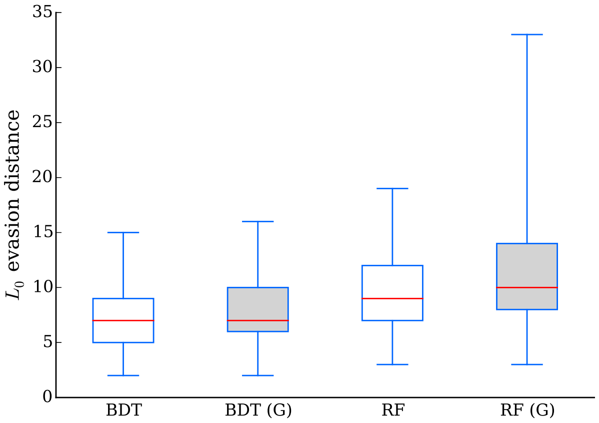


Figure 6.4: Optimal and heuristic evasion bounds on the evaluation dataset. The heuristic bounds are both denoted by "(G)" and grayed.

6.6 Discussion

The results presented in this chapter bring a set of sometimes astonishing observations regarding the behavior of off-the-shelf machine learning classifiers under evasion attacks. Our first observation is that by and large, current machine learning techniques present large blind spots: for any correctly classified instance, there exists a variety of close-by evading instances which the classifier gets wrong. This phenomenon largely occurs across model classes and happens in spite of the extremely high testing accuracies of the classifiers. Even though the evading instances are highly non-random and the specific targeted distortions must be carefully computed, this is points to a major deficiency in the generalization power of machine learning. None of the tested classifiers has actually learned the concept of a handwritten “2” or “6”, but has rather compactly memorized salient features of the training set.

Tree ensembles perform worse in this regard than other classifiers, even against linear classifiers, the simplest of all model classes. Both random forests and boosted trees exhibit high sensitivity to adversarial L_ρ perturbations, particularly for $\rho \in \{1, 2\}$. This is perhaps unsurprising. The predicates of the decision nodes perform hard cuts on feature values, so that for a feature value already close to the threshold, a tiny push in the right direction results in taking the opposite branch. Note that this instability is not a problem when operating on binary valued features only, and this might explain why tree ensembles fare better against L_0-evasions. This “hard cut” property might also play a role in the extreme weakness of the L_0-hardened ensemble against all other types of evasions. At any rate, the non-linearity of tree ensembles does not immunize them against evasions.

Another significant observation is the disappointing performance of the large-margin approach embodied by the convex polytope model. The evasion susceptibilities of the CPM and neural network models are almost indistinguishable, even though the model classes are quite different. Although trivially related to evasion (un)susceptibility, the large-margin idea does not seem to significantly help. One possible explanation is that large-margin is a training time constraint, but evasion operates at testing time. In other words, a classifier that has a large separation margin on a training dataset might not enjoy the same margin on random testing instances. Another explanation is that our relaxation of the max-margin problem, which forms the basis of the training algorithm, somehow denatures the large-margin requirement.

All is not lost however. Adversarial boosting is surprisingly effective at “patching” the blind spots of tree ensembles. The adversarially boosted classifier has a slightly higher accuracy than the regularly boosted model, and enjoys a five fold increase in L_0-evasion robustness. Adversarial boosting seems to operate like a powerful regularization method, forcing the model to take into account significantly more evidence over the feature values before reaching a decision. The bottom right picture in Figure 6.1 shows an evenly spread and attenuated decision importance of the pixels for the hardened model. In contrast, the regular model strongly focuses on a small number of features that are powerful discriminants for the training set, but useless in the presence of a testing time adversary. Unfortunately, the hardened model exhibits considerable sensitivity to all other types of evading adversaries. Whether tree ensembles can be adversarially boosted against all types of L_ρ deformations remains an open question.

Chapter 7

Conclusion

In taming evasion attacks against machine learning based detection pipelines, we encounter a rich set of ideas, both in terms of practical and theoretical challenges. We now take a step back and shed some light on important conceptual observations about the task of designing robust detection pipelines.

In Chapter 3, we introduce a novel feature extraction for detecting comment spam. In contrast to previous approaches, our feature extractor is less susceptible to simple evasion attacks based on word spelling variations and better captures the essence of comment spam. Generally speaking, if the feature extraction ψ loses task-relevant information in the embedding process, no amount of subsequent machine learning can defend against evasion attacks. This particular situation is characterized by the existence of malicious and benign observations $\omega^+ \neq \omega^-$ having identical, or equivalently, arbitrarily close feature vectors $\psi(\omega^+) = \psi(\omega^-)$.

Hence, when designing a robust machine learning based detection pipeline, we should always begin by paying close attention to the feature extraction step. In many cases however, determining the list of exhaustive traits of the problem space Ω which captures the complete semantics of the task, and should thus be embedded, is impractical. Even if such a list could be made, we can expect some important measurements to be extremely expensive depending on our level of access to Ω. For example, in the comment spam case study, we extensively leverage the vantage point offered by the data provider to correlate messages across tens of thousands of websites. Had we operated at the level of a single website however, no such information would have been available to us, and our obtained groupings would have mostly consisted in useless singletons.

From this discussion, it would seem that to be evasion-safe, a good feature extractor should embed as much information as possible about the observation space Ω. After all, the task of the subsequent machine learning algorithms is to sift through the data and automatically weed-out the non-relevant noise features, focussing on the signal alone. Unfortunately, current machine learning methods do not behave in this way.

Statistical significance is one potential issue. Growing the size of the feature space provides more opportunity for spurious, coincidental patterns to show up, which in turn implies that larger amounts of training data are required to avoid overfitting. For the Convex Polytope Machine, but also for most of the machine learning algorithms, the generalization bounds scale in $O(n^{-\frac{1}{2}})$.

However, because of the remarkable fact that generalization bounds do not depend on the number of features d but instead on the complexity of the model class, one can still achieve good classification accuracy even for very large values of d. Modern distributed and parallel implementations of learning algorithms can handle large amounts of instances n, so that choosing a small-enough model class that also correctly fits the training data is the key to overcome the issue of statistical significance.

Having set statistical significance aside, we argue that the main issue faced by off-the-shelf machine learning in those circumstances is the familiar correlation/causation confusion. By breaking the tacit assumption that training and testing data are generated by the same process, evasion attacks cast a hard light on the inability of learning algorithms to differentiate between patterns that are merely *correlated* with a given label, and patterns that actually *cause* or *explain* the label. In part II, and specifically in Chapter 6, our evasion algorithms show that common classifiers systematically fail to consider all the available evidence in support of a given decision. This makes evasion particularly easy by changing a few key features that happen to be statistically correlated to the label in the training set. For the handwritten digit recognition task, those features fall short of capturing the underlying semantics of the pictures.

The adversarial boosting framework proposes a cure to this state of affairs. Conceptually, adversarial boosting is a powerful interactive regularization technique. At every step, the classifier is inspected to reveal what it has learned. Practically, the classifier is asked to generate instances which it "thinks" are representative of one label or another. An oracle-like mechanism then provides the ground truth labels of the generated instances, so that the classifier's beliefs can be corrected by retraining. Although the actual implementation of adversarial boosting is not described in those terms, this presentation of adversarial boosting is isomorphic to its introductive definition in Section 5.6. In particular, the evasion algorithm is nothing less than a general purpose classifier inversion technique: an evasion method finds an instance x such that $f(x) = 1$ or $f(x) = -1$ with additional constraints on the form of x. Because these constraints are of the form "x is not too far away from another instance x' in the training set", we know what the true label for the generated instance x should be, namely the label of x'. Hence, the evasion algorithm is the part that inspects f by inversion, and the labeling oracle part is accomplished by the boundedness constraint on $\Delta(x, x')$.

The main obstacle to the hardening of practical security decision pipelines by adversarial boosting is the case-by-case design of an adapted distance function Δ. This function essentially defines the set of label-invariant transformations that are allowed in the application domain. Δ does not have to be exhaustive for adversarial boosting to work. It only has to capture plausible input transformations that an evading adversary may try. In a fundamental way, Δ and more largely adversarial boosting provide the opportunity for the machine learning practitioner to customize the learning algorithm for the specific requirements of the application domain while still enjoying much of the benefits of off-the-shelf machine learning techniques.

Bibliography

Barreno, M., B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar (2006). "Can Machine Learning Be Secure?" In: *Proceedings of the ACM Symposium on Information, Computer and Communications Security*. ASIACCS '06, pp. 16–25.

Barron, A. R., J. Rissanen, and B. Yu (1998). "The Minimum Description Length Principle in Coding and Modeling". In: *IEEE Transactions on Information Theory* 44.6, pp. 2743–2760.

Bartlett, P. L., M. I. Jordan, and J. D. McAuliffe (2003). "Large Margin Classifiers: Convex Loss, Low Noise, and Convergence Rates." In: *Proceedings of the Neural Information Processing Systems Conference*. NIPS '03, pp. 1173–1180.

Bartlett, P. L. and S. Mendelson (2003). "Rademacher and Gaussian Complexities: Risk Bounds and Structural Results". In: *Journal of Machine Learning Research* 3, pp. 463–482.

Baum, L. E. and T. Petrie (1966). "Statistical Inference for Probabilistic Functions of Finite State Markov Chains". In: *The Annals of Mathematical Statistics* 37.6, pp. 1554–1563.

Belotti, P., P. Bonami, M. Fischetti, A. Lodi, M. Monaci, A. Nogales-Gomez, and D. Salvagnin (2016). "On Handling Indicator Constraints in Mixed Integer Programming". In: *Computational Optimization and Applications*, pp. 1–22.

Benenson, R., M. Mathias, R. Timofte, and L. Van Gool (2012). "Pedestrian Detection at 100 Frames per Second". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. CVPR '12. IEEE, pp. 2903–2910.

Bergstra, J., O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio (2010). "Theano: a CPU and GPU Math Expression Compiler". In: *Proceedings of the Python for Scientific Computing Conference*. SciPy '10.

Beygelzimer, A., J. Langford, Y. Lifshits, G. Sorkin, and A. Strehl (2009). "Conditional Probability Tree Estimation Analysis and Algorithms". In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. UAI '09, pp. 51–58.

Beygelzimer, A., J. Langford, and B. Zadrozny (2005). "Weighted One-against-all". In: *Proceedings of the National Conference on Artificial Intelligence*. AAAI '05, pp. 720–725.

Biggio, B., I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli (2013). "Evasion Attacks against Machine Learning at Test Time". In: *Machine Learning and Knowledge Discovery in Databases*. Vol. 8190. Lecture Notes in Computer Science. Springer.

Boser, B. E., I. M. Guyon, and V. Vapnik (1992). "A Training Algorithm for Optimal Margin Classifiers". In: *Proceedings of the Annual Workshop on Computational Learning Theory*. COLT '92. New York, NY, USA: ACM, pp. 144–152.

Bottou, L. (2010). "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *Proceedings of COMPSTAT'2010: 19th International Conference on Computational Statistics*, pp. 177–186.

— (2012). "Neural Networks: Tricks of the Trade: Second Edition". In: ed. by G. Montavon, G. B. Orr, and K.-R. Muller. Springer. Chap. Stochastic Gradient Descent Tricks, pp. 421–436.

Boyd, S. and L. Vandenberghe (2004). *Convex Optimization*. Cambridge University Press.

Bratko, A., B. Filipic, G. V. Cormack, T. R. Lynam, and B. Zupan (2006). "Spam Filtering Using Statistical Data Compression Models". In: *Journal of Machine Learning Research* 7, pp. 2673–2698.

Breiman, L. (1996). "Bagging predictors". In: *Machine learning* 24.2, pp. 123–140.

— (2001). "Random Forests". In: *Machine Learning* 45.1, pp. 5–32.

Brennan, M., S. Wrazien, and R. Greenstadt (2010). "Learning to Extract Quality Discourse in Online Communities". In: *Proceedings for the AAAI Conference on Collaboratively-Built Knowledge Sources and Artificial Intelligence*. AAAIWS' 10, pp. 4–9.

Broder, A. Z., M. Charikar, A. M. Frieze, and M. Mitzenmacher (1998). "Min-wise Independent Permutations". In: *Journal of Computer and System Sciences* 60, pp. 327–336.

Bruckner, M., C. Kanzow, and T. Scheffer (2012). "Static Prediction Games for Adversarial Learning Problems". In: *Journal of Machine Learning Research* 13.1, pp. 2617–2654.

Buitinck, L., G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux (2013). "API Design for Machine Learning Software: Experiences from the Scikit-Learn Project". In: *Proceedings of the ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122.

Carlini, N., P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou (2016). "Hidden Voice Commands". In: *Proceedings of the USENIX Security Symposium*. USENIX Security '16. To appear.

Carlini, N. and D. Wagner (2016). *Defensive Distillation is Not Robust to Adversarial Examples*. To appear.

Chang, C. and C. Lin (2011). "LIBSVM: A library for support vector machines". In: *ACM Transactions on Intelligent Systems and Technology* 2 (3).

Chen, B.-C., J. Guo, B. Tseng, and J. Yang (2011). "User Reputation in a Comment Rating Environment". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '11.

Chen, T. and T. He (2014). *XGBoost: eXtreme Gradient Boosting*. `https://github.com/dmlc/xgboost`. Accessed: 2016-06-15.

Chen, T. and C. Guestrin (2016). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the ACM SIGKKD International Conference on Knowledge Discovery and Data Mining*. KDD '16.

Cleary, J. and I. Witten (1984). "Data Compression Using Adaptive Coding and Partial String Matching". In: *IEEE Transactions on Communications* 32.4, pp. 396–402.

Cortes, C. and V. Vapnik (1995). "Support-Vector Networks". In: *Machine Learning* 20.3, pp. 273–297.

Dalvi, N., P. Domingos, Mausam, S. Sanghai, and D. Verma (2004). "Adversarial classification". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '04, pp. 99–108.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 1–38.

Dietterich, T. G. and G. Bakiri (1995). "Solving Multiclass Learning Problems via Error-correcting Output Codes". In: *Journal of Artificial Intelligence Research* 2.1, pp. 263–286.

Djuric, N., L. Lan, S. Vucetic, and Z. Wang (2013). "BudgetedSVM: A Toolbox for Scalable SVM Approximations". In: *Journal of Machine Learning Research* 14, pp. 3813–3817.

Dredze, M., K. Crammer, and F. Pereira (2008). "Confidence-Weighted Linear Classification". In: *Proceedings of the International Conference on Machine Learning*. ICML '08. ACM, pp. 264–271.

Erhan, D., Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio (2010). "Why does Unsupervised Pre-training Help Deep Learning?" In: *Journal of Machine Learning Research* 11, pp. 625–660.

Fan, R.-E. (2011). *LibSVM Datasets*. `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`. [Online; accessed 10-June-2016].

Fan, R.-E., K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin (2008). "LIBLINEAR: A Library for Large Linear Classification". In: *Journal of Machine Learning Research* 9, pp. 1871–1874.

Felzenszwalb, P. F., R. B. Girshick, D. McAllester, and D. Ramanan (2010). "Object Detection with Discriminatively Trained Part Based Models". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.9, pp. 1627–1645.

Fischer, P. (1995). "More or Less Efficient Agnostic Learning of Convex Polygons". In: *Proceedings of the Conference on Computational Learning Theory*. COLT '95. ACM, pp. 337–344.

Freund, Y. and R. E. Schapire (1995). "A Decision-Theoretic Generalization of Online Learning and an Application to Boosting". In: *Proceedings of the Conference on Computational Learning Theory*. COLT '95. Springer, pp. 23–37.

Freund, Y. and R. E. Schapire (1999). "Large Margin Classification Using the Perceptron Algorithm". In: *Machine Learning* 37.3, pp. 277–296.

Freund, Y., R. E. Schapire, and N. Abe (1999). "A Short Introduction to Boosting". In: *Japanese Society For Artificial Intelligence* 14.771-780, p. 1612.

Friedman, J. H. (2001). "Greedy Function Approximation: A Gradient Boosting Machine." In: *The Annals of Statistics* 29.5, pp. 1189–1232.

Friedman, J. H., T. Hastie, R. Tibshirani, et al. (2000). "Additive Logistic Regression: a Statistical View of Boosting". In: *The Annals of Statistics* 28.2, pp. 337–407.

Gao, H., J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Zhao (2010). "Detecting and Characterizing Social Spam Campaigns". In: *Proceedings of the Internet Measurement Conference*. ICM '10.

Glorot, X., A. Bordes, and Y. Bengio (2011). "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the International Conference on Artificial Intelligence and Statistics*. AISTATS '11, pp. 315–323.

Goodfellow, I. J., J. Shlens, and C. Szegedy (2015). “Explaining and Harnessing Adversarial Examples”. In: *Proceedings of the International Conference on Learning Representations*. ICLR ’15.

Graves, A., A.-R. Mohamed, and G. E. Hinton (2013). “Speech Recognition with Deep Recurrent Neural Networks”. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. ICASSP ’13. IEEE, pp. 6645–6649.

Gu, S. and L. Rigazio (2015). “Towards Deep Neural Network Architectures Robust to Adversarial Examples”. In: *arXiv preprint*. arXiv: `arXiv:1412.7063v4`.

Gurobi Optimization, I. (2016). *Gurobi Optimizer Reference Manual*. [Online; accessed 15-June-2016].

Hao, S., A. Kantchelian, B. Miller, N. Feamster, and V. Paxson (2016). “PREDATOR: Proactive Recognition and Elimination of Domain Abuse at Time-Of-Registration”. In: *Proceedings of the SIGSAC Conference on Computer and Communications Security*. CCS ’16. To appear. ACM.

Hastie, T., R. Tibshirani, J. Friedman, and J. Franklin (2005). “The Elements of Statistical Learning: Data Mining, Inference and Prediction”. In: 27.2, pp. 83–85.

Hinton, G. E. (2012). “A practical guide to training restricted Boltzmann machines”. In: *Neural Networks: Tricks of the Trade*, pp. 599–619.

Hinton, G. E., O. Vinyals, and J. Dean (2014). “Distilling the knowledge in a neural network”. In: *Proceedings of the Deep Learning and Representation Learning Workshop at NIPS*.

Hoad, T. C. and J. Zobel (2003). “Methods for Identifying Versioned and Plagiarized Documents”. In: *Journal of the American Society for Information Science and Technology* 54.3, pp. 203–215.

Honglak, L. and A. Ng (2005). “Spam Deobfuscation Using a Hidden Markov Model”. In: *Proceedings of the Conference on Email and Anti-Spam*. CEAS ’05.

Hornik, K. (1991). “Approximation Capabilities of Multilayer Feedforward Networks”. In: *Neural Networks* 4.2, pp. 251–257.

Howard, M. and S. Lipner (2009). *The security development lifecycle*. O’Reilly Media, Incorporated.

Hsu, C.-F., E. Khabiri, and J. Caverlee (2009). “Ranking Comments on the Social Web”. In: *Proceedings of the IEEE International Conference on Computational Science and Engineering*. CSE ’09, pp. 90–97.

Isacenkova, J., O. Thonnard, A. Costin, A. Francillon, and D. Balzarotti (2014). “Inside the Scam Jungle: a Closer Look at 419 Scam Email Operations”. In: *EURASIP Journal on Information Security* 2014.1, pp. 1–18.

Kantchelian, A. (2014). *Convex Polytope Machine Implementation*. `https://github.com/alkant/cpm`. [Online; accessed 10-June-2016].

Kantchelian, A., S. Afroz, L. Huang, A. C. Islam, B. Miller, M. C. Tschantz, R. Greenstadt, A. D. Joseph, and J. D. Tygar (2013). “Approaches to Adversarial Drift”. In: *Proceedings of the Artificial Intelligence and Security Workshop*. AISec ’13. ACM, pp. 99–110.

Kantchelian, A., J. Ma, L. Huang, S. Afroz, A. Joseph, and J. D. Tygar (2012). “Robust Detection of Comment Spam Using Entropy Rate”. In: *Proceedings of the Security and Artificial Intelligence Workshop*. AISec ’12. ACM, pp. 59–70.

Kantchelian, A., M. C. Tschantz, L. Huang, P. L. Bartlett, A. D. Joseph, and J. D. Tygar (2014). “Large-Margin Convex Polytope Machine”. In: *Proceedings of the Neural Information Processing Systems Conference*. NIPS ’14.

Kantchelian, A., M. C. Tschantz, S. Afroz, B. Miller, V. Shankar, R. Bachwani, A. D. Joseph, and J. D. Tygar (2015). “Better Malware Ground Truth: Techniques for Weighting Anti-Virus Vendor Labels”. In: *Proceedings of the Artificial Intelligence and Security Workshop*. AISec ’15. ACM, pp. 45–56.

Kantchelian, A., J. D. Tygar, and A. Joseph (2016). “Evasion and Hardening of Tree Ensemble Classifiers”. In: *Proceedings of the International Conference on Machine Learning*. ICML ’16.

Kearns, M. J. and Y. Mansour (1998). “A Fast, Bottom-Up Decision Tree Pruning Algorithm with Near-Optimal Generalization.” In: *Proceedings of the International Conference on Machine Learning*. Vol. 98. ICML ’98, pp. 269–277.

Kerckhoffs, A. (1883). “La Cryptographie Militaire”. In: *Journal des Sciences Militaires* 9, p. 538.

King, J. C. (1976). “Symbolic Execution and Program Testing”. In: *Communications of the ACM* 19.7, pp. 385–394.

Kohavi, R. (1996). “Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid.” In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Vol. 96. KDD ’96, pp. 202–207.

LeCun, Y., L. Bottou, G. B. Orr, and K.-R. Muller (2012). “Efficient Backprop”. In: *Neural Networks: Tricks of the Trade*. Springer, pp. 9–48.

LeCun, Y., C. Cortes, and C. J. Burges (1998). *MNIST dataset*.

Lee, K., J. Caverlee, and S. Webb (2010). “Uncovering Social Spammers: Social Honeypots + Machine Learning”. In: *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’10.

Levchenko, K., A. Pitsillidis, N. Chachra, B. Enright, M. Felegyhazi, C. Grier, T. Halvorson, C. Kanich, C. Kreibich, H. Liu, et al. (2011). “Click trajectories: End-to-end analysis of the spam value chain”. In: *Proceedings of the IEEE Symposium on Security and Privacy*. SP ’11. IEEE, pp. 431–446.

Levenberg, K. (1944). “A Method for the Solution of Certain Non-Linear Problems in Least Squares”. In: *Quarterly of Applied Mathematics* 2, pp. 164–168.

Lewis, D. D., Y. Yang, T. G. Rose, and F. Li (2004). “RCV1: A New Benchmark Collection for Text Categorization Research”. In: *Journal of Machine Learning Research* 5, pp. 361–397.

Li, P. (2012). “Robust LogitBoost and Adaptive Base Class (ABC) LogitBoost”. In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. UAI ’12.

Loosli, G., S. Canu, and L. Bottou (2007). “Training Invariant Support Vector Machines using Selective Sampling”. In: *Large Scale Kernel Machines*. MIT Press, pp. 301–320.

Lowd, D. and C. Meek (2005a). “Adversarial Learning”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. KDD ’05.

— (2005b). “Good Word Attacks on Statistical Spam Filters”. In: *Proceedings of the Conference on Email and Anti-Spam*. CEAS ’05.

Ma, J., L. K. Saul, S. Savage, and G. M. Voelker (2009). “Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’09, pp. 1245–1254.

Manwani, N. and P. S. Sastry (2010). “Learning Polyhedral Classifiers Using Logistic Function”. In: *Proceedings of the Asian Conference on Machine Learning*. ACML ’10, pp. 17–30.

— (2013). “Polyceptron: A Polyhedral Learning Algorithm”. In: *arXiv:1107.1564*.

Marquardt, D. W. (1963). “An Algorithm for Least-Squares Estimation of Nonlinear Parameters”. In: *Journal of the Society for Industrial and Applied Mathematics*. Vol. 11. 2, pp. 430–441.

McCoy, D., A. Pitsillidis, J. Grant, N. Weaver, C. Kreibich, B. Krebs, G. Voelker, S. Savage, and K. Levchenko (2012). “Pharmaleaks: Understanding the business of online pharmaceutical affiliate programs”. In: *Proceedings of the USENIX Security Symposium*. USENIX Security ’12, pp. 1–16.

McCullagh, P. and J. A. Nelder (1989). *Generalized Linear Models*. Chapman and Hall/CRC.

Mehta, M., J. Rissanen, and R. Agrawal (1995). “MDL-Based Decision Tree Pruning”. In: *Proceedings of the International Conference on Knowledge Discovery and Data Mining*. Vol. 21. KDD ’95 2, pp. 216–221.

Miller, B., A. Kantchelian, S. Afroz, R. Bachwani, R. Faizullabhoy, L. Huang, V. Shankar, T. M. C., T. Wu, G. Yiu, A. Joseph, and J. D. Tygar (2016). “Reviewer Integration and Performance Measurement for Malware Detection”. In: *Proceedings of the Conference on Detection of Intrusions, Malware & Vulnerability Assessment*. DIMVA ’16. Springer.

Miller, B., A. Kantchelian, S. Afroz, R. Bachwani, E. Dauber, L. Huang, M. C. Tschantz, A. D. Joseph, and J. D. Tygar (2014). “Adversarial Active Learning”. In: *Proceedings of the Artificial Intelligence and Security Workshop*. AISec ’14. ACM, pp. 3–14.

Mishne, G., D. Carmel, and R. Lempel (2005). “Blocking Blog Spam with Language Model Disagreement”. In: *Proceedings of the International Workshop on Adversarial Information Retrieval on the Web*. AIRWeb ’05.

Mishne, G. and N. Glance (2006). “Leave a Reply: An Analysis of Weblog Comments”. In: *Proceedings of the International World Wide Web Conference*. WWW ’06.

Mishra, A. and R. Rastogi (2012). “Semi-Supervised Correction of Biased Comment Ratings”. In: *Proceedings of the International World Wide Web Conference*. WWW ’12.

Nelson, B., B. I. P. Rubinstein, L. Huang, A. D. Joseph, S. J. Lee, S. Rao, and J. D. Tygar (2012). “Query Strategies for Evading Convex-Inducing Classifiers”. In: *Journal of Machine Learning Research* 13.

Norouzi, M., M. Collins, M. A. Johnson, D. J. Fleet, and P. Kohli (2015). “Efficient non-greedy optimization of decision trees”. In: *Proceedings of the Neural Information Processing Systems Conference*. NIPS ’15, pp. 1720–1728.

Papernot, N., P. McDaniel, X. Wu, S. Jha, and A. Swami (2015). “Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks”. In: *Proceedings of the IEEE Symposium on Security and Privacy*. SP ’15.

Pavlov, I. (2007). *LZMA SDK (Software Development Kit)*.

Peters, J. and S. Schaal (2008). “Reinforcement Learning of Motor Skills with Policy Gradients”. In: *Neural Networks* 21.4. Robotics and Neuroscience, pp. 682–697.

Prokhorov, D. (2001). “The IJCNN 2001 Neural Network Competition”. In: *Slide presentation in the International Joint Conference on Neural Networks (IJCNN)*.

Ramachandran, A. and N. Feamster (2006). “Understanding the Network-Level Behavior of Spammers”. In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM ’06.

Ramachandran, A., N. Feamster, and S. Vempala (2007). “Filtering Spam with Behavioral Blacklisting”. In: *Proceedings of the SIGSAC Conference on Computer and Communications Security*. CCS ’07. ACM.

Raykar, V. C., S. Yu, L. H. Zhao, G. H. Valadez, C. Florin, L. Bogoni, and L. Moy (2010). “Learning From Crowds”. In: *Journal of Machine Learning Research* 11, pp. 1297–1322.

Richard H. Byrd, P. L., J. Nocedal, and C. Zhu (1995). “A Limited Memory Algorithm for Bound Constrained Optimization”. In: *SIAM Journal on Scientific Computing* 16.5, pp. 1190–1208.

Russell, S. J. and P. Norvig (2009). *Artificial Intelligence: A Modern Approach*. 3rd ed. Pearson Education.

Schurmann, T. and P. Grassberger (1996). “Entropy Estimation of Symbol Sequence”. In: *Chaos* 6.3, pp. 414–427.

Sculley, D. and G. M. Wachman (2007). “Relaxed Online SVMs for Spam Filtering”. In: *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’07, pp. 415–422.

Shalev-Shwartz, S., Y. Singer, and N. Srebro (2007). “Pegasos: Primal Estimated sub-GrAdient SOlver for SVM”. In: *Proceedings of the International Conference on Machine Learning*. ICML ’07, pp. 807–814.

Shannon, C. E. (1949). “Communication Theory of Secrecy Systems”. In: *Bell System Technical Journal* 28.4, pp. 656–715.

— (1951). “Prediction and Entropy of Printed English”. In: *Bell System Technical Journal* 30.1, pp. 50–64.

Shin, Y., M. Gupta, and S. Myers (2011). “Prevalence and Mitigation of Forum Spamming”. In: *Proceedings of the IEEE International Conference on Computer Communications*. INFOCOM ’11. IEEE, pp. 2309–2317.

Smola, A. J., P. L. Bartlett, B. Scholkopf, and D. Schuurmans, eds. (2000). *Advances in Large Margin Classifiers*. MIT press.

Smutz, C. and A. Stavrou (2012). “Malicious PDF Detection Using Metadata and Structural Features”. In: *Proceedings of the Annual Computer Security Applications Conference*. ACSAC ’12. ACM, pp. 239–248.

— (2016). “When a Tree Falls: Using Diversity in Ensemble Classifiers to Identify Evasion in Malware Detectors”. In: *Proceedings of the Network and Distributed Systems Security Symposium*. NDSS ’16.

Srivastava, N., G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Srndic, N. and P. Laskov (2014). “Practical Evasion of a Learning-Based Classifier: A Case Study”. In: *Proceedings of the IEEE Symposium on Security and Privacy*. SP ’14.

Stamper, J., A. Nicolescu-Mizil, S. Ritter, G. J. Gordon, and K. R. Koedinger (2010). *Algebra I 2008-2009. Challenge data set from KDD Cup 2010 Educational Data Mining Challenge*.

Sutskever, I., O. Vinyals, and Q. V. Le (2014). "Sequence to Sequence Learning with Neural Networks". In: *Proceedings of the Neural Information Processing Systems Conference*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. NIPS '14, pp. 3104–3112.

Szegedy, C., W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus (2013). "Intriguing Properties of Neural Networks". In: *Proceedings of the International Conference on Learning Representations*. ICLR '13.

Takacs, G. (2010). "Smooth Maximum Based Algorithms for Classification, Regression, and Collaborative Filtering". In: *Acta Technica Jaurinensis* 3.1, pp. 27–63.

Thomas Hofmann Bernhard Scholkopf, A. J. S. (2008). "Kernel Methods in Machine Learning". In: *The Annals of Statistics* 36.3, pp. 1171–1220.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag.

Vinyals, O., A. Toshev, S. Bengio, and D. Erhan (2015). "Show and Tell: A Neural Image Caption Generator". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. CVPR '15, pp. 3156–3164.

Wang, Z., N. Djuric, K. Crammer, and S. Vucetic (2011). "Trading Representability for Scalability: Adaptive Multi-Hyperplane Machine for Nonlinear Classification". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '11, pp. 24–32.

Wolsey, L. A. (2007). "Mixed Integer Programming". In: *Wiley Encyclopedia of Computer Science and Engineering*. John Wiley & Sons, Inc.

Xie, Y., F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov (2008). "Spamming Botnets: Signatures and Characteristics". In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '08. ACM, pp. 171–182.

Xu, J.-M., G. Fumera, F. Roli, and Z.-H. Zhou (2009). "Training SpamAssassin with Active Semi-supervised Learning". In: *Proceedings of the Conference on Email and Anti-Spam*. CEAS '09.

Xu, W., Y. Qi, and D. Evans (2016). "Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers". In: *Proceedings of the Network and Distributed Systems Security Symposium*. NDSS '16.

Zhu, Z. A., W. Chen, G. Wang, C. Zhu, and Z. Chen (2009). "P-packSVM: Parallel primal gradient descent kernel SVM". In: *Proceedings of the International Conference on Data Mining*. ICDM '09. IEEE, pp. 677–686.

Zhuang, L., J. Dunagan, D. R. Simon, H. J. Wang, I. Osipkov, G. Hulten, and J. D. Tygar (2008). "Characterizing Botnets from Email Spam Records". In: *Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats*. LEET '08.