UNIVERSITY OF CALIFORNIA, SAN DIEGO

Learning to Detect Malicious URLs

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

in

Computer Science

by

Justin Tung Ma

Committee in charge:

Professor Lawrence K. Saul, Co-Chair
Professor Stefan Savage, Co-Chair
Professor Geoffrey M. Voelker, Co-Chair
Professor Pamela Cosman
Professor Gert Lanckriet

2010

The dissertation of Justin Tung Ma is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____
Co-Chair

_____
Co-Chair

_____
Co-Chair

University of California, San Diego

2010

DEDICATION

To Symantec, where peculiar circumstances steered my research in a bright new direction.

EPIGRAPH

千里之行，始于足下
—老子

A journey of a thousand miles begins with a single step.
—Lao Zi

(It is reassuring that even in ancient times, procrastination was a problem.)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

In the first email I wrote to one of my advisors, I nervously began with "Dear Dr. Savage," hoping that I had complied with the strict protocol for addressing superiors the academic hierarchy. His response surprised me: "That's what they call my father. You can call me Stefan. :)"

From then on I knew things would change — *I* would change. Over the years, the warped expectations I held an undergraduate would evolve into a more confident, patient and accepting outlook on life. My time at UC San Diego was a transforming experience, and I have many people to thank for being a part of it.

First and foremost, I want to thank my advisors: Professors Stefan Savage, Geoff Voelker and Lawrence Saul. Their lessons shaped who I am as a researcher, a communicator, and a citizen.

To them, research had to be fun, or else it became a rote technical exercise. I also learned research was a chance to solve important problems that affected the lives of millions of people. To keep things in perspective, I learned to remember that "why" we do things was just as important as "how."

My advisors give fun talks, and their writing is enjoyable to read. Over the years I have tried my best to emulate their example. To them, it was more important to communicate clearly than to show how smart you were. To that end, I am thankful for the grueling practice talk sessions and insightful paper revisions where they dragged me out of my bubble and reminded me to see things from the audience's perspective.

To them, being a good citizen was paramount. From paper reviews to interacting with other researchers, my advisors would always remind me: "How would you feel if you were on the other side?" To that end, I've learned to be more open-minded and patient — to see outside of my own bubble and appreciate what others have to offer.

Finally, I have always admired my advisors' efforts to enrich the department community, which have included funding radical Chez Bob projects, running classes on how to make video games, teaching classes on how to write effectively, and participating in holiday skits. Efforts like these go a long way in making the

department a more welcoming place. That spirit and desire to promote the community inspired me to participate in holiday skits, to coordinate the "Half-Baked Half Hour" and "How to Get a Job" seminars, and to join in other social activities.

I am grateful for the camaraderie of a number of department colleagues: Michael Vrable, James Anderson, Alvin AuYoung, Kirill Levchenko, Qing Zhang, Chris Kanich, Ryan Braud, Marti Motoyama, Patrick Verkaik, Alden King, Kaisen Lin, Anshuman Gupta, Andreas Pitsillidis, Damon McCoy, Nathan Farrington, John McCullough, Gjergji Zyba, Diane Hu, Chih-Chieh Cheng, Laurens van der Maaten, Shibin Parameswaran, Brian McFee, Kai Wang, Samory Kpotufe, Yuvraj Agarwal, Harsha Madhyastha, and Professors Alex Snoeren and Amin Vahdat. I am also grateful for the camaraderie of a number of SysNet alumni: Barath Raghavan, Diwaker Gupta, Shaan Mahbubani, Chris Tuttle, Ranjita Baghwan, Renata Teixeira, David Moore, Kashi Vishwanath, Priya Mahadevan, Dejan Kostic (and his epic clapping), Chip Killian (and his futile allegiance to Papa Johns), Ramana Kompella, Marvin McNett, Jeanne Albrecht, Jay Chen, and Erik Vandekieft. All are amazing people from whom I learned an immeasurable amount.

I also had the opportunity to play intramural sports alongside Alvin, as well as ECE alumni Brendan Morrison and Arun Batra. I am especially thankful I still have my teeth!

I had some great collaborations with people outside of UCSD. Thanks to Christian Kreibich and Mark Felegyhazi from ICSI, as well as the UPenn gang: Alex Kulesza, Mark Dredze, Koby Crammer and Fernando Pereira. Working with these guys was a blast.

Michael Frederick, Johannes Hewel and Adam Mclean were my housemates during my first year. I thank them for the good times (which include watching Lost and Battlestar Galactica) and helping me adjust to San Diego.

During my stay, I picked up a few new hobbies and had the pleasure of meeting many interesting people along the way. From Argentine Tango, I want to thank Mila Khoroosi, Reza Farsian, Vlad Spasojevic, Quinn Read, Maziar Nezhad, Rachel McRoberts, Chris Calabro, Paola Robotti, Sabah and Zeina Chammas, Lampis Zalavras, Robin Thomas, Chris X. Edwards, Murat and Michelle Erdemsel,

Homer and Cristina Ladas, Natasha Vayner, Chi Essary, Richard Essary, Eva Contreras, and Liz Gire. From Ashtanga Yoga, I want to thank my instructors Natasha Teran and Michelle Hackett, as well as fellow yogi/PhD student Kourtney Murray. From Archery, I want to thank Thomas Pham, Lloyd Brown, Ruth Rowe for their instruction, and the Andrews brothers (Larry and Torin).

Wushu had been a big part of my life since I was 10 years old. Experiencing the sport as an athlete, coach, judge and organizer was a rare and illuminating experience. I thank Sujal Bista, Zhibo Lai and the whole Terp Wushu Club for their camaraderie and support through the years. Special thanks to Anita Lopez, long-time teammate and fellow tournament organizer, without whom I would have gone crazy running the Team Trials. And special thanks to USAWKF president Anthony Goh, who (against tremendous opposition) gave me the opportunity to run the 2006 and 2007 Team Trials. During that time, I learned a number of invaluable lessons about being an organizer and navigating politics. Although I am no longer involved in the sport, I will keep those lessons with me as I navigate my new careers.

Classical piano has been a big part of my life since I was 8 years old. I stopped practicing after my 1st year as an undergraduate but resumed during the 4th year in graduate school. Music played a pivotal role in keeping me balanced in my final PhD years, and I thank Ruben Pelaez and Laura Garcia de Mendoza for being my teachers. With each passing day, I grow more thankful for their gift. I also thank Scott Laird for the opportunity to be a soloist in multiple, unforgettable concerto concerts.

The following professors and instructors at Maryland played pivotal roles in my development as an undergraduate student and were instrumental in encouraging me to attend graduate school: Michelle Hugue, Bobby Bhattacharjee, Bill Arbaugh, Jonathan Katz, Ray St. Leger, James Yorke, Art Eckstein and Jeannie Rutenberg.

I had a few tight situations during graduate school. I want to thank Brian Kantor for getting me out of a jam back in my 1st year; Julie Conner and Kathryn Krane for getting me out of yet another jam in my 3rd year. Thank you all for

your patience.

CSE would be nothing without its wonderful staff. Among others, I want to thank Michelle Panik, Kim Graves, Paul Terry, Yuka Nakanishi, Jocelyn Bernardo, Samira Khazai and Anne Goldenberg.

Tremendous thanks to ECE professors Pamela Cosman and Gert Lanckriet for taking time to serve on my thesis committee.

Finally, but most importantly, I thank my family for their love and support.

Chapters 1, 2 and 3, in part, are reprints of the material as it appears in Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) 2009. Ma, Justin; Saul, Lawrence K.; Savage, Stefan; Voelker, Geoffrey M. The dissertation author was the primary investigator and author of this paper.

Chapters 1 and 4, in part, are reprints of the material as it appears in Proceedings of the International Conference on Machine Learning (ICML) 2009. Ma, Justin; Saul, Lawrence K.; Savage, Stefan; Voelker, Geoffrey M. The dissertation author was the primary investigator and author of this paper.

Chapter 6, in part, is a reprint of the material as it appears in Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS) 2010. Ma, Justin; Kulesza, Alex; Dredze, Mark; Crammer, Koby; Saul, Lawrence K.; Pereira, Fernando. The dissertation author was a primary investigator and author of this paper.

VITA AND PUBLICATIONS

2004	B. S. in Computer Science *summa cum laude*, University of Maryland, College park
2004	B. S. in Mathematics *summa cum laude*, University of Maryland, College park
2007	M. S. in Computer Science, University of California, San Diego
2010	Ph. D. in Computer Science, University of California, San Diego

Justin Ma, Alex Kulesza, Mark Dredze, Koby Crammer, Lawrence K. Saul, and Fernando Pereira, "Exploiting Feature Covariance in High-Dimensional Online Learning," *Proc. of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, Sardinia, Italy, May 2010.

Qing Zhang, John McCullough, Justin Ma, Nabil Schear, Michael Vrable, Amin Vahdat, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage, "Neon: System Support for Derived Data Management," *Proc. of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, Pittsburgh, PA, March 2010.

Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker, "Identifying Suspicious URLs: An Application of Large-Scale Online Learning," *Proc. of the International Conference on Machine Learning (ICML)*, pages 681–688, Montreal, Quebec, June 2009.

Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker, "Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs," *Proc. of the ACM SIGKDD Conference*, pages, 1245–1253, Paris, France, June 2009.

Justin Ma, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey M. Voelker, "Unexpected Means of Protocol Inference," *Proc. of the ACM Internet Measurement Conference (IMC)*, Rio de Janeiro, Brazil, October 2006.

Justin Ma, John Dunagan, Helen J. Wang, Stefan Savage, and Geoffrey M. Voelker, "Finding Diversity in Remote Code Injection Exploits," *Proc. of the ACM Internet Measurement Conference (IMC)*, Rio de Janeiro, Brazil, October 2006.

Justin Ma, Geoffrey M. Voelker, and Stefan Savage, "Self-Stopping Worms," *Proc. of the ACM Workshop on Rapid Malcode (WORM)*, Fairfax, VA, November 2005.

Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage, "Scalability, Fidelity and Containment in the Potemkin Virtual Honeyfarm," *Proc. of the 20th ACM Symposium on Operating System Principles (SOSP)*, Brighton, UK, October, 2005.

Minho Shin, Justin Ma, Arunesh Mishra, and William A. Arbaugh, "Wireless Security and Interworking," *Proc. of the IEEE*, Vol. 94, No. 2, February, 2006.

ABSTRACT OF THE DISSERTATION

Learning to Detect Malicious URLs

by

Justin Tung Ma

Doctor of Philosophy in Computer Science

University of California, San Diego, 2010

Professor Lawrence K. Saul, Co-Chair
Professor Stefan Savage, Co-Chair
Professor Geoffrey M. Voelker, Co-Chair

Malicious Web sites are a cornerstone of Internet criminal activities. They host a variety of unwanted content ranging from spam-advertised products, to phishing sites, to dangerous "drive-by" exploits that infect a visitor's machine with malware. As a result, there has been broad interest in developing systems to prevent the end user from visiting such sites. The most prominent existing approaches to the malicious URL problem are manually-constructed blacklists, as well as client-side systems that analyze the content or behavior of a Web site as it is visited.

The premise of this dissertation is that we should be able to construct a lightweight URL classification system that simultaneously overcomes the challenges that face blacklists (which have manual updates that can quickly become obsolete) and client-side systems (which are difficult to deploy on a large scale because of their high overhead). To this end, our contribution is that we develop a highly effective system for malicious URL detection that (in its final form) leverages large

numbers of features and online learning to scalably and adaptively construct an accurate classifier. Because our system exploits large amounts of training data and adapts to day-by-day variations, we are able to classify URLs with up to 99% accuracy.

As part of pursuing malicious URL detection, this dissertation addresses issues that arise from the use of online learning for this application. Thus, our further contributions include advances in understanding the role of uncertainty in online learning, as well as the benefits of exploiting feature correlations in high-dimensional applications such as URL classification. Overall, the contributions of this dissertation make significant advances in improving malicious URL detection and understanding the role of online learning in this application.

Chapter 1

Introduction

From an early age we learn to manage personal risk: to infer which situations may be dangerous and avoid them accordingly. For example, most cities have a "rough part of town" that is understood to be more dangerous for casual visitors. However, this same notion translates poorly to the Internet context — there are few effective "rules of thumb" to differentiate safe Web sites from those that are dangerous. Indeed, Internet criminals *depend* on the absence of such indicators to prey on their marks.

Thus, the Web has become a platform for supporting a wide range of criminal enterprises such as spam-advertised commerce (e.g., counterfeit watches or pharmaceuticals), financial fraud (e.g., via phishing or 419-type scams) and malware propagation (e.g., trojan downloads and so-called "drive-by exploits"). Although the precise commercial motivations behind these schemes may differ, the common thread among them is the requirement that unsuspecting users visit their sites. These visits can be driven by email, Web search results or links from other Web pages, but all require the user to take some action, such as clicking, that specifies the desired Uniform Resource Locator (URL).

Clearly, if one could inform users *beforehand* that a particular URL was dangerous to visit, much of this problem could be alleviated. To this end, the security community has responded by developing blacklisting services — encapsulated in toolbars, appliances and search engines — that provide precisely this feedback. These blacklists are in turn constructed by a range of techniques includ-

ing manual reporting, honeypots, and Web crawlers combined with site analysis heuristics. Inevitably, many malicious sites are not blacklisted — either because they are too new, were never evaluated, or were evaluated incorrectly (e.g., due to "cloaking"). To address this problem, some client-side systems analyze the content or behavior of a Web site *as* it is visited. But, in addition to run-time overhead, these approaches can expose the user to the very browser-based vulnerabilities that we seek to avoid.

In this dissertation, we focus on a complementary part of the design space: *lightweight* URL classification — that is, classifying the reputation of a Web site *entirely* based on the inbound URL. The motivation is to provide inherently better coverage than blacklisting-based approaches (e.g., correctly predicting the status of new sites) while avoiding the client-side overhead and risk of approaches that analyze Web content on demand. In the following, we summarize this dissertation's contributions to the problem of detecting malicious URLs.

1.1 Contributions

This dissertation's addresses the problem of detecting malicious Web sites using machine learning over URL-based features. We want to simultaneously overcome the issues that plague blacklists (which have manual updates that cannot keep up with newly-introduced malicious sites) and client-side systems (which are difficult to deploy on a large scale because of their high overhead). The challenge is to construct a system that is accurate, scalable and adaptive. To this end, the contributions of this dissertation are the following:

- We demonstrate the effectiveness of using a large number of features for automated, malicious URL detection (Chapter 3).
- We use online learning to scalably and adaptively tackle the problem of large-scale, real-time malicious URL detection (Chapter 4).

We also make contributions to the field of online learning that arise from our investigation of large-scale malicious URL detection:

- A side-by-side comparison of Bayesian logistic regression and confidence-weighted learning (Chapter 5).
- The development of low-rank methods for approximating the full covariance in high-dimensional online learning (Chapter 6).

As a result, we show that it is feasible to construct a scalable and adaptive system for malicious URL detection that is highly accurate, and we deepen our understanding of the benefits of online learning in large-scale applications.

In the following, we provide an overview of the chapters of the dissertation and elaborate on our contributions.

1.2 Organization

Chapter 2 covers essential background material for the dissertation and discusses related work.

Chapter 3 describes our initial feasibility study on using machine learning for malicious URL detection. In this work, we face the challenge of sifting through myriad URL-based features to automatically construct a viable URL classifier. In contrast to previous work, we successfully demonstrate our approach of using a large number of lexical and host-based features for classification. Given these encouraging results, we conduct a follow-on study over more realistic, large-scale data.

Chapter 4 presents our work of detecting malicious URLs over a large-scale, real-time feed of URLs. The challenge for this work is to exploit large amounts of training data (on the order of millions of examples) and to adapt to changes in malicious URLs over time (since new data arrives on a continual basis). We address the problem by building a system that collects URL features in real time and employs online learning to construct an accurate and timely classifier. Confidence-weighted (CW) learning [DCP08, CDP09] is the most successful algorithm in this study because it treats features differently — learning more aggressively for feature weights with uncertain estimates, and learning less aggressively for feature weights with confident estimates. CW's success inspires follow-on work comparing CW to other

online algorithms on our application (Chapter 5), as well as work on augmenting CW to exploit feature correlations in high-dimensional spaces (Chapter 6).

Chapter 5 compares confidence-weighted learning with Bayesian logistic regression (BLR), two online algorithms that incorporate per-feature-weight uncertainty in ways that are very similar yet fundamentally different. Through carefully constructed synthetic experiments and through evaluations on URL classification, we highlight the advantages and disadvantages of BLR vs. CW, giving practitioners insight on when to choose one algorithm over the other. In particular, we show that CW is more suited to malicious URL detection than BLR because CW adapts more quickly to newly-introduced features. This adaptiveness is important for successfully learning a classifier on the evolving data streams that characterize this application.

Chapter 6 explores the benefits and tradeoffs of exploiting feature covariances in high-dimensional spaces for online learning algorithms such as CW. Up to this point, we have relied on modeling the uncertainty of feature weights independently — i.e., using a diagonal covariance matrix. In this chapter, we use synthetic experiments to demonstrate when using a full covariance matrix is beneficial. Then, we develop a low-rank, factored approximation of the full covariance matrix for high-dimensional applications — such as malicious URL detection — for which using a full covariance is computationally infeasible.

Finally, Chapter 7 presents conclusions of the dissertation and discusses future work.

Chapter 1, in part, is a reprint of the material as it appears in Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) 2009. Ma, Justin; Saul, Lawrence K.; Savage, Stefan; Voelker, Geoffrey M. The dissertation author was the primary investigator and author of this paper.

Chapter 1, in part, is a reprint of the material as it appears in Proceedings of the International Conference on Machine Learning (ICML) 2009. Ma, Justin; Saul, Lawrence K.; Savage, Stefan; Voelker, Geoffrey M. The dissertation author was the primary investigator and author of this paper.

Chapter 2

Background and Related Work

The World Wide Web is a global information network that users can access through the Internet, and this network consists of a collection of Web sites. An individual Web site is a collection of related text pages, videos, images and other resources that are hosted on a Web server. Typically, users access Web sites through browsers, client software that fetches and renders the text, images and other content associated with a site (examples of popular contemporary browser programs are Firefox, Internet Explorer, Chrome and Safari). However, the browser must *locate* the desired site before fetching, and Uniform Resource Locators (URLs) are the standard way of naming locations on the Web [BLFM98, Voe99].

2.1 What is a URL?

Just as we use filenames to locate files on a local computer, we use Uniform Resource Locators (URLs) to locate Web sites (as well as individual Web resources). One way users visit a site is by typing a URL into the browser's address bar. An arguably easier way is to click a link, which is contained within a page that is already rendered by the browser or an email message rendered by a mail client. In either case, the link contains the URL of the desired site. Because sites often link to other sites, the network of links among Web sites would be similar to a spider web (hence the naming of the Web).

However, URLs are text strings that are human readable but not directly

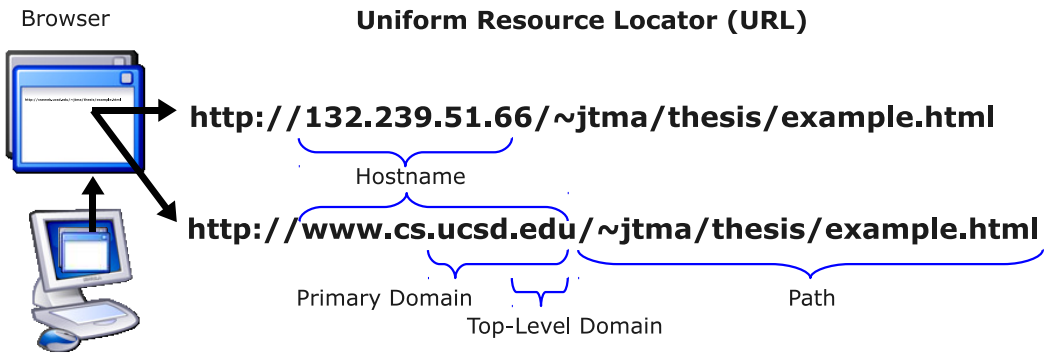


Figure 2.1: Example of a Uniform Resource Locator (URL) and its components.

usable by client programs. Through a multi-step resolution process, the browser translates a URL into instructions on how to locate the server that is hosting the site, as well as where within that host the site or resource is placed. Because a URL must go through this machine translation process, it has the following standard syntax:

```
<protocol>://<hostname><path>
```

The `<protocol>` portion of the URL indicates which network protocol should be used to fetch the requested resource. The most common protocols in use are Hypertext Transport Protocol or HTTP (`http`), HTTP with Transport Layer Security (`https`) and File Transfer Protocol (`ftp`). In Figure 2.1, all of the example URLs specify the HTTP protocol. Although we do not include the protocol as a classification feature (most of the URLs we encounter in our training sources use `http`), we observe that phishing exploits often insert tokens such as `http` into the URL's path to trick the user into thinking that the URL is legitimate. For example, if we own `maliciousssite.com` but want to trick the user into thinking they are visiting `www.cs.ucsd.edu`, then we may construct a URL such as `http://maliciousssite.com/http:/www.cs.ucsd.edu`.

The `<path>` of a URL is analogous to the path name of a file on a local computer. In Figure 2.1, the path in the example is `/~jtma/thesis/example.html`. The path tokens, delimited by various punctuation such as slashes, dots and dashes, show how the site is organized. However, criminals sometimes obscure path tokens to avoid scrutiny, or they may deliberately construct tokens to mimic the appearance of a legitimate site, as in the case of phishing (see Chapter 3.3.6 for examples of such tokens).

The `<hostname>` is the identifier for the Web server on the Internet. Sometimes it is a machine-readable Internet Protocol (IP) address, but more often (especially from the user's perspective) it is a human-readable domain name.

IP is a network protocol that makes it possible for a host to send network packets to another host on the Internet, regardless of the underlying networking hardware. A key principle behind IP is that all hosts on the Internet have an IP address and can thus reach one another. In IP version 4 (IPv4) [USC81], addresses are 32-bit integers that are typically represented as a dotted quad. In dotted quad notation, we divide the 32-bit address into four 8-bit bytes. Then, from most to least significant byte, we print each byte in decimal notation delimited by dots (e.g., 132.239.51.66 in Figure 2.1). This ordering of the bytes is significant because it lets us represent hierarchies in the address space using IP prefix notation (Section 2.4.3). Other representations such as hexadecimal, dotted quad of hexadecimal numbers, dotted quad of octal numbers and binary are interchangeable with the dotted quad of decimal numbers we described. However, we restrict our attention to the dotted quad of decimal numbers because it is the most popular representation. Finally, we restrict our attention to IPv4 because the newer IP version 6 [DH98] (with 128-bit addresses) is not as widely deployed at the moment.

Because hostnames represented as IP addresses do not convey high-level information about the host (such as which organization it belongs to), we typically use human-readable domain names instead. A domain name is a series of text tokens delimited by dots '.'; they describe the hostname's place in a multi-level naming hierarchy. The right-most token indicates the domain name's major categorization — e.g., `.com` and `.edu` for generic domain names, and `.cn`

and `.uk` for international domain names. Moving from right-to-left, each token identifies the hostname at progressively lower levels of the hierarchy (known as "subdomains") and provides a blueprint for translating the domain name into an IP address. For example, the domain `www.cs.ucsd.edu` from Figure 2.1 represents a host (`www.cs.ucsd.edu`) contained within the computer science department (`cs.ucsd.edu`) at UC San Diego (`ucsd.edu`), whose domain registration is handled by EDUCAUSE (the `.edu` registrar). To better understand the mechanisms behind the domain name, we describe how a name is translated into an IP address (Section 2.4.1) and how a name is established (Section 2.4.2).

We have described URLs at a high level in this section and provide further background on the mechanisms for constructing, resolving and hosting a Web site's URL in Section 2.4. But before we can understand the implications of URL resolution mechanics on the ability to detect malicious Web sites, we need a high-level overview of malicious sites themselves.

2.2 Malicious Web Sites

Malicious Web sites encompass a range of different criminal enterprises. Although the precise motivations may differ among the criminals who construct these sites, malicious sites present a danger to users who unsuspectingly visit them. Nevertheless, because different genres of malicious sites present different threats to users, it is helpful to examine these genres individually to arrive at a better understanding of their commonalities. In doing so, we can develop an intuition for the kinds of defenses that will work against all genres instead of tailoring solutions to a particular kind of threat.

There are four major categories of malicious sites, and each category is distinguished by the level of interaction required by the user, as well as the motivation of the site owners.

- **Illicit Products:** These sites typically sell illicit goods such as pharmaceuticals (often counterfeit), counterfeit watches and pirated software. Because of the criminal nature of these enterprises, these sites typically rely on URLs

(a) Illicit Products (b) Phishing

(c) Trojan Downloads

Figure 2.2: Examples of three categories of malicious Web sites: (a) illicit products, (b) phishing, and (c) trojan downloads.

in spam advertising or artificially-optimized Web search results to attract traffic. Figure 2.2(a) shows an example of a pharmacy campaign selling male enhancement drugs.

- **Phishing:** Phishing sites use mimicry to trick users into divulging private information such as passwords and account numbers. First, phishers send email messages to unsuspecting users that prompt the users to visit the phishing site's URL. Users are then asked to enter their sensitive information on the site, which is intentionally crafted to resemble a legitimate site (e.g., their bank, email or social networking account page). A common deception phishers use is to embed tokens from the legitimate target site into their own phishing URL (e.g., `http://www.bankofamerica.com.maliciousssite.com`).

After the criminals acquire the user's credentials, they may use them to withdraw money, launder money, send spam under a new guise, as well as commit other nefarious acts. We present a phishing example in Figure 2.2(b), which compares the legitimate login page for Facebook with the illegitimate login page for the phishing site `fblight.com`.

- **Trojan Downloads:** These sites encourage users to download and execute a malware binary that purports to be benign. Figure 2.2(c) shows an example of a site that encourages users to download an electronic postcard. Other examples of trojan downloads can include sites hosting screensavers, codec plugins for video players, and other seemingly mundane programs. Criminal organizations construct these kinds of sites to infect new hosts and expand their networks of compromised hosts (known as botnets).

- **Drive-By Exploits:** Drive-by exploits are scripts or objects embedded in a Web page that exploit a vulnerability in the user's browser. Once the browser is compromised, the exploit code causes the user to involuntarily download malicious code. Examples of drive-by exploits include maliciously-crafted javascript, Flash objects and images. Criminals may embed the exploits into stand-alone pages, but often the exploits are also loaded into legitimate sites via third-party advertising. Although we do not directly address the threat of drive-by exploits loaded as advertisements (this would require analyzing content), our approach could still address this problem by letting legitimate servers determine the safety of the advertisement URLs before including such third-party content.

Although the motivations differ behind these genres of malicious sites, there are common properties related to their URLs that we can leverage. Because some of these properties are related to the mechanics of URL resolution, we reserve a more detailed discussion for Section 2.4 but allude to salient properties here to provide intuition.

In particular, because these sites are taken down regularly as part of enforcement efforts, we can view the recentness of their new WHOIS domain registrations

with suspicion. Moreover, these take-down efforts prompt criminals to quickly rotate through new Web servers. As a result, criminals will often host these sites on compromised user machines or machines hosted by disreputable service providers (e.g., "bullet-proof" hosting providers who deliberately ignore victims' complaints about the abuse perpetrated by their criminal customers). Moreover, criminals will often use a similarly illicit hosting configuration for their domain name system (DNS) infrastructure, which is responsible for directing users to the site. Finally, the textual makeup of the URLs themselves (particularly for phishing sites) can provide clues about malicious intent.

Finally, because malicious Web sites represent a significant threat to users, an early (and still prominent) solution to the problem is to blacklist those sites.

2.3 Blacklists

In the Web browsing context, blacklists are precompiled lists (or databases) that contain IP addresses, domain names or URLs of malicious sites that users should avoid. (By contrast, whitelists contain sites that are known to be safe.)

To cross reference a site against a blacklist, users submit a given IP address, domain name or URL to the blacklist's query service. In return, they receive a response for whether the site is in the blacklist. The most popular implementations of blacklist query services are domain name system-based (DNS-based) blacklists, browser toolbars and network appliances.

2.3.1 DNS-Based Blacklists

DNS blacklists are a query service implemented on top of DNS (we describe the details of DNS in Section 2.4.1). Users submit a query (representing the IP address or the domain name in question) to the blacklist provider's special DNS server, and the response is an IP address that represents whether the query was present in the blacklist.

SORBS, URIBL, SURBL and Spamhaus are examples of major DNS blacklist providers. To get a concrete idea of how these lists operate, we provide an ex-

ample from URIBL. If we encounter a site `unknown.com`, then performing a DNS lookup on `unknown.com.multi.uribl.com` could return the address `127.0.0.2` (which represents membership in their blacklist) or `host not found` (which means the site is not on the blacklist). The mechanism is similar for other major DNS blacklist providers, although the exact implementation details differ.

The most prominent uses for DNS blacklists are filtering the IP addresses of spam message senders, as well as marking messages with URLs that contain blacklisted domain names. Although generic Web browsing is not the most prominent use case for DNS blacklists, the intertwined nature of spam sending and malicious sites makes DNS blacklists useful features for a URL classification system (see Section 3.1.1).

2.3.2 Browser Toolbars

Browser toolbars provide a client-side defense for users. Before a user visits a site, the toolbar intercepts the URL from the address bar and cross references a URL blacklist, which is often stored locally on the user's machine or on a server that the browser can query. If the URL is present on the blacklist, then the browser redirects the user to a special warning screen that provides information about the threat. The user can then decide whether to continue to the site.

McAfee SiteAdvisor, Google Toolbar and WOT Web of Trust are prominent examples of blacklist-backed browser toolbars. McAfee's SiteAdvisor is backed by a blacklist of URLs collected by virtual machine-based Web crawlers (which test for exploits and scams); SiteAdvisor also accepts user feedback and analyzes links to and from sites [McA]. WOT Web of Trust is a URL blacklist that is constructed collaboratively through user contributions and feedback [Aga]. The Google Toolbar is a browser plugin that queries a Google-maintained URL blacklist [Goo]. The Google blacklist itself is constructed using a large-scale machine learning-based approach that is similar to ours [WRN10]; their approach is contemporary work that was published subsequent to our work in Chapters 3–4.

2.3.3 Network Appliances

Dedicated network hardware is another popular option for deploying blacklists. These appliances serve as proxies between user machines within an enterprise network and the rest of the Internet. As users within an organization visit sites, the appliance intercepts outgoing connections and cross references URLs or IP addresses against a precompiled blacklist. The operating principle is similar to browser toolbars, but the positioning of the appliance at the network gateway saves the overhead of installing special software on all machines within the organization.

IronPort and WebSense are examples of companies that produce blacklist-backed network appliances. Before constructing the blacklist, IronPort Web Reputation assigns reputations to URLs based on a weighted score of several features including URL category, IP address information, and domain registration information [Iro08]. Likewise, the WebSense ThreatSeeker Network assigns reputations to URLs and IP addresses before constructing the final blacklist [Web].

2.3.4 Limitations

The primary advantage of blacklists is that querying is a low overhead operation: the lists of malicious sites are precompiled, so the only computational cost of deployed blacklists is the lookup overhead. However, the need to construct these lists in advance gives rise to their most glaring disadvantage: blacklists become stale.

Blacklist construction is a perpetual game of "catch up." Network administrators block existing malicious sites, and enforcement efforts take down criminal enterprises behind those sites. There is a constant pressure on criminals to construct new sites and to find new hosting infrastructure. As a result, new malicious URLs are introduced and blacklist providers must update their lists yet again. However, in this vicious cycle, criminals are always ahead because Web site construction is inexpensive: domain registration is cheap (on the order of $10 per year at major registrars such as GoDaddy), subdomains of a registered domain can be generated for free (which is useful if blacklists record the entire domain name instead of the primary domain), and Web site templates are reusable. Moreover, free

services for blogs (e.g., Blogger) and personal hosting (e.g., Google Sites, Microsoft Live Spaces) provide another inexpensive source of disposable sites.

In light of blacklist limitations, a machine learning approach to detecting malicious URLs would be a promising step toward breaking this vicious cycle. For this alternative approach to succeed, we would have to anticipate trends in the various properties (features) related to malicious URLs. As we will see in the next section, many useful features arise from URL resolution mechanisms.

2.4 Mechanics of URL Resolution

We provide background on the mechanisms involved in constructing, resolving and hosting a Web site's URL. We relate these mechanics to the goal of detecting malicious URLs, hinting at their applications as useful indicators (features) for automated detection and deferring a detailed discussion about the features to Chapter 3.1.1.

2.4.1 Domain Name System Resolution

The Domain Name System (DNS) is a hierarchical network of servers responsible for translating domain names into IP addresses and other kinds of information [Moc87a, Moc87b]. During the DNS resolution process, the tokens in the domain name are traversed from right-to-left to direct the client's DNS resolver to query the appropriate DNS name servers.

Let us take the hostname `www.cs.ucsd.edu` from Figure 2.1 as an example. First, the resolver will query a DNS root server to resolve the `.edu` portion of the domain name. The response from the root server will be a name server (NS) record that directs the resolver to the `.edu` name server. Next, the resolver will query the `.edu` name server for the records corresponding to `ucsd.edu`; in return the resolver receives the address for the `ucsd.edu` name server. Then, a request to the `ucsd.edu` name server for `cs.ucsd.edu` would return the NS record pointing to the `cs.ucsd.edu` name server. Finally a query to the `cs.ucsd.edu` name server with a request containing `www.cs.ucsd.edu` would return an address (A) record

containing the IP address of `www.cs.ucsd.edu` (which in this case is 132.239.51.66). Alternatively, we could query the `cs.ucsd.edu` name server for the mail exchanger (MX) record containing the IP address of the mail server associated with the `cs.ucsd.edu` domain.

The A, MX and NS records are IP addresses of hosts associated with a domain name. Under the hypothesis that the hosting infrastructure for malicious URLs is distinct from that for benign URLs, the various DNS records become useful differentiators. The A records for malicious Web servers might be hosted on compromised residential machines or in disreputable providers. The NS records would represent the DNS infrastructure that leads a victim to a malicious site — this infrastructure is also likely to be hosted on compromised machines or disreputable providers. Moreover, if the set of hosts responsible for hosting malicious sites are affiliated with the hosts that send spam for a mailing campaign, then the associated MX records would be reliable indicators of malice. The stories of the McColo, Troyak and Group 3 hosting providers reinforce the notion that malicious sites tend to be hosted in bad places:

- McColo provided hosting for major botnets, which in turn were responsible for sending 41% of the world's spam just before McColo's takedown in November 2008 [Kei08].

- Troyak and Group 3 were responsible for hosting 90 out of 249 command-and-control servers for the Zeus botnet before their takedown in March 2010 [McM08].

On a related note, there is a special DNS record type called the pointer (PTR) record. Its purpose is to enable reverse DNS lookups: given an IP address as a query, the associated domain name is returned. To perform a reverse lookup on `132.239.51.66`, a client submits a DNS query for the domain name `66.51.239.132.in-addr.arpa` (note the reverse byte order of the IP address) and receives `www.cs.ucsd.edu` as the PTR record response. The existence of a PTR record is a reliable indicator that the domain name is well-established — as such, the PTR record is a potentially useful classification feature.

Finally, although A, MX and NS records show promise as classification features, individual IP addresses may be too fine-grain for characterizing malicious hosting activity. To address this problem, we describe the standard for representing an aggregate set of IP addresses in Section 2.4.3. Next, we cover how the binding between name server and domain name is established.

2.4.2 Domain Name Registration

Besides the IP addresses associated with a domain name, there is useful information associated with domain name registration.

Registration establishes which name servers are associated with a domain name. Typically, the registrant registers the primary domain name (a term we define shortly) with the registrar; the registrant is the owner of the domain name, and the registrar is the organization responsible for hosting the NS record that points to the primary domain's servers.

The top-level domain (TLD) is the right-most token in a domain name — e.g., `.com`, `.edu`, `.cn`, `.uk`. A registrar is usually in charge of a single TLD, although it is possible for a registrar to delegate that authority to other smaller registrars. The primary domain is the highest level domain name that a registrant can register. It usually consists of the two right-most tokens in the domain (e.g., `google.com`), although it may be the three right-most tokens in international domain names (e.g., `google.co.uk`). In the Figure 2.1 example, `.edu` is the TLD and `ucsd.edu` is the primary domain.

The information associated with a domain name registration can indicate whether certain entities have a higher tendency of registering domains associated with malicious sites, as well as whether a site is newly registered and has yet to establish its credibility. This information includes vital data about the registrant, the registrar, date of registration, date of expiration, date of the latest update, and other attributes associated with the record; it is typically stored in databases accessible through the WHOIS query/response protocol [Dai04]. WHOIS works as follows:

1. The client contacts the registrar's WHOIS server on TCP port 43.

2. The registrar returns a plaintext response, or redirects the query to a smaller registrar that is in charge of the designated WHOIS record.

Because the WHOIS server infrastructure and plaintext response format is ad-hoc compared to DNS, the query time may take on the order of seconds (especially for queries to registrars with less infrastructure). As a result, some registrars put a quota on the number of WHOIS queries that a user can perform in a day. Thus, if we are concerned about query overhead and rate limiting, we may choose to omit WHOIS features. In fact, evaluations in Section 3.3.2 show that although the inclusion of WHOIS data yields the highest accuracy, the exclusion of WHOIS data still results in a highly accurate classifier.

2.4.3 Routing and Forwarding

As we alluded in Section 2.4.1, individual IP addresses are very fine-grain because there are more than 4 billion possible addresses; recording enough of them to characterize the location of malicious URLs can be overwhelming. Thus, it is beneficial for us to represent an IP address as belonging to a block of IP addresses. This more coarse-grain view of representing addresses is important because individual hosts can have address churn (e.g., because of the Dynamic Host Configuration Protocol [Dro97]). Moreover, we may be able to identify aggregate behaviors if certain activities tend to occur in specific subsections of the Internet (see the McColo example in Section 2.4.1). Fortunately, we can use existing representations from Internet routing and forwarding for these address blocks.

Routing is the process of setting up paths along which IP packets are forwarded. Internet service providers (ISPs) negotiate these paths using the border gateway protocol (BGP). In BGP, ISPs are also known as autonomous systems (ASes), which are identified by their AS numbers. Thus, we can treat AS numbers as *de facto* identifiers for ISPs: if an ISP is responsible for hosting a collection of malicious sites, then we can use the AS number as a salient feature (e.g., McColo's AS number was 26780).

Similarly, we can also treat IP prefixes as *de facto* identifiers for ISPs. During packet forwarding, routers must redirect incoming packets by looking up the

next-hop router responsible for forwarding to the packet's destination address. Storing individual IP addresses in the forwarding table would consume a prohibitive amount of memory, especially for routers at the Internet core. However, routers save space by referencing groups of IP addresses using classless interdomain routing (CIDR) aggregation [FL06]. An IP prefix (or CIDR block) is a compact way of denoting a block of IP addresses. In Internet Protocol version 4 (IPv4), the notation `<address>/<prefix_len>` represents all IP addresses that share the `<prefix_len>` most significant bits with `<address>`. For example, `137.110.222.0/24` represents all IP addresses in the range `137.110.222.0`–`137.110.222.255`. Typically, authorities assign IP prefixes to ISPs, thereby making these prefixes useful identifiers of providers (for McColo, it was 208.66.192.0/22).

We note that the mapping from AS numbers to IP prefixes is not necessarily isomorphic. It is possible for an AS to contain multiple IP prefix blocks, or for an IP prefix block to be spread across multiple ASes. Thus, because AS numbers provide a slightly different granularity than IP prefixes, both sets of features are complementary — we include both in our evaluations. As we will see in Section 3.3.2, the addition of AS numbers and IP prefixes (along with other host-based properties) results in highly accurate classification results.

Next, we provide an overview of related work in the area of detecting malicious URLs to place this dissertation in context.

2.5 Related Work

In this dissertation, we describe our approach to detecting malicious Web sites using machine learning over the lexical and host-based features of their URLs. The previous section provided technical background for understanding our approach to constructing features in Chapters 3 and 4. In the following, we place this dissertation in context with related work in URL classification using learning methods, the use of statistical methods in related applications, as well as other approaches to URL classification. Because our techniques are based on lexical and host-based features of the URL, we place more emphasis in surveying related work

that incorporates those features.

2.5.1 Classification of URLs

Researchers have sought opportunities to move beyond conventional means to address the problem of identifying suspicious URLs. Previous work has either employed batch machine learning techniques or performed extensive feature analysis without using machine learning. The work in this dissertation differs in that we employ a more comprehensive feature set and address the problem of scalable learning over large data sets.

Kan and Thi [KT05] provide one of the early efforts of classifying URLs using machine learning. Because they want to train their models quickly, they only use lexical features — i.e., analyzing the URL string and ignoring properties related to the page content and the host. They use a bag-of-words representation of tokens in the URL with annotations about the tokens' positions within the URL (e.g., hostname, path, etc.). They also use consecutive n-grams of tokens, ordered bigrams of tokens that do not have to be consecutive, and the lengths of different parts of the URL. A noteworthy result is that lexical features can achieve 95% of the accuracy of page content features. We use a similar but simpler set of lexical features in this dissertation, yet we are able to achieve high accuracy with them (see "Lexical" results in Section 3.3.2).

The work by Garera et al. is the most closely related to ours [GPCR07]. They use logistic regression over 18 hand-selected features to classify phishing URLs. The features include the presence of red flag key words in the URL, features based on Google's Page Rank, and Google's Web page quality guidelines. Although a direct comparison with our approach is difficult without access to the same URLs or features, they achieve a classification accuracy of 97.3% over a set of 2,500 URLs. Though similar in motivation and methodology, our approach differs significantly in both scope (aiming to detect other types of malicious activity) and scale (considering orders-of-magnitude more features and training examples).

McGrath and Gupta do not construct a classifier but nevertheless perform a comparative analysis of phishing and non-phishing URLs [MG08]. With respect

to data sets, they compare non-phishing URLs drawn from the DMOZ Open Directory Project [Net] to phishing URLs from PhishTank [Ope] and a non-public source. The features they analyze include IP addresses, WHOIS thin records (containing date and registrar-provided information only), geographic information, and lexical features of the URL (length, character distribution, and presence of predefined brand names). We build on their work by incorporating similar sources and features into our approach.

Provos et al. perform a study of drive-by exploit URLs and use a patented machine learning algorithm as a pre-filter for virtual machine (VM) based analysis [PMRM08]. Unlike our approach, they extract content-based features from the page, including whether inline frames are "out of place" (an "IFrame" is a window within a page that can contain another page), the presence of obfuscated javascript, and whether IFrames point to known exploit sites. In their evaluations, the ML-based pre-filter can achieve 0.1% false positives and 10% false negatives. Although a direct comparison is not possible because of different data sets, our evaluations in Section 3.3.4 yielded similar performance despite the exclusion of content.

CANTINA classifies phishing URLs by thresholding a weighted sum of 8 features (4 content-related, 3 lexical, and 1 WHOIS-related) [ZHC07]. Among the lexical features, it looks at dots in the URL, whether certain characters are present, and whether the URL contains an IP address. The WHOIS-related feature CANTINA examines is the age of the domain. These features are a subset of those we use in our approach, but we use more sophisticated classification models. Although a direct comparison is not possible, we show that more sophisticated classification models achieve higher accuracy than simple ones (e.g., logistic regression and support vector machines perform better than the simple Naive Bayes in Section 3.3.3).

Guan et al. [GCL09] focus on classifying URLs that appear in instant messaging (IM). Although they use several URL-based features, they also take advantage of a number of IM-specific features such as message timing and content. Among the URL-based features are the age of the domain (WHOIS lookup), Google

rank and a number of lexical features (IP address in the hostname, presence of particular tokens in the URL, as well as the length of certain hostname tokens). They use an ad-hoc linear classifier where the weight of each feature is proportional to the difference in the number of benign examples that possess the feature and the number of malicious examples that do not. As we explained in the discussion of CANTINA, we believe a more sophisticated classification model would yield higher accuracy.

2.5.2 Machine Learning and URL Features in Related Contexts

The work in this dissertation deals with classifying URLs in a generic setting, irrespective of the applications in which they appear. However, there is a body of related work that uses URL-based features for classifying email messages and Web pages (not the URLs themselves).

Fette et al. use statistical methods in machine learning to classify phishing messages [FST07]. Their classifiers examine the properties of URLs contained within a message (e.g., the number of URLs, number of domains, and number of dots in a URL), but unlike our approach they also consider features of the email structure and content. Bergholz et al. further improve the accuracy of Fette et al. by introducing models of text classification to analyze email content [BCP+08]. Abu-Nimeh et al. compare different classifiers over a corpus of phishing messages, using as features the frequency of the 43 most-popular words in the corpus [ANNWN07].

Kolari et al. use URLs found within a blog page as features to determine whether the page is spam [KFJ06]. They use a "bag-of-words" representation of URL tokens, and we use a similar set of features in our approach which contribute to a highly accurate classifier (see Section 3.3.2).

2.5.3 Non-Machine Learning Approaches

Highly Predictive Blacklists (HPBs) address the limitations of traditional blacklists by allowing participants to construct blacklists in a distributed fashion [ZPU08]. In this way, HPBs bypass the inefficiencies associated with centralized blacklist construction. Although the focus of HPB is network-level intrusion detection, HPBs introduce a notion of cooperative, community-based labeling that is similar in spirit to future work we propose in Section 7.2.2.

Several projects have also explored operating systems-level techniques where the client visits the Web site using an instrumented virtual machine (VM). The VM can emulate any client-side exploits that occur as a result of visiting a malicious site, and the instrumentation can detect whether an infection has occurred. In this way, the VM serves as a protective buffer for the user. Moshchuk et al. use VMs to analyze downloaded trojan executables, relying on third-party anti-spyware tools to detect whether VMs were infected by executables [MBGL06]. Wang et al. detect drive-by exploits by using behavioral detection (monitoring anomalous state changes in the VM) as well as detecting exploits of known vulnerabilities [WBJ+06]. Provos et al. monitor VM state changes and use multiple anti-virus engines to detect VM infection [PMRM08]. Moshchuk et al. also construct a VM-based Web proxy defense that uses behavior-based detection and adds only a few seconds of overhead to page rendering for the end-client [MBD+07]. These efforts have distinct benefits (direct detection of certain classes of sites such as drive-by exploits) and limitations (inadvertently exposing the user to undetected drive-by exploits and other malicious sites that do not fit their precise detection criteria); thus, they are complementary to our approach.

Chapter 2, in part, is a reprint of the material as it appears in Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) 2009. Ma, Justin; Saul, Lawrence K.; Savage, Stefan; Voelker, Geoffrey M. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Moving Beyond Blacklists

Malicious Web sites are a cornerstone of Internet criminal activities. As a result, there has been broad interest in developing systems to prevent the end user from visiting such sites. To that end, our goal is to construct a system for detecting malicious URLs that is both scalable (for learning over large-scale data sets) and adaptive (especially to changes in malicious URLs over time). The current chapter presents the first step of our approach: investigating the effectiveness of our features using batch classification over smaller-scale data (on the order of 10^4 examples). Later in Chapter 4, we will adopt online learning techniques to scale our approach to large-scale data sets (on the order of 10^6 examples).

In contrast to blacklists (Section 2.3), our approach to the problem is based on automated URL classification. In particular, we explore the use of statistical methods from machine learning for classifying site reputation based on the relationship between URLs and the lexical and host-based features that characterize them. Unlike previous work, we show that these methods are able to sift through tens of thousands of features derived from publicly available data sources, and can identify which URL components and meta-data are important without requiring heavy domain expertise. Indeed, our system automatically selects many of the same features identified by domain experts as being typical of "malicious" Web sites. But more importantly, our system selects new, non-obvious features that are highly predictive and yield substantial performance improvements. We evaluate the feasibility of this approach across 20,000 to 30,000 URLs drawn from different

sources, and show that it can obtain an overall prediction accuracy of 95–99%, detecting a large fraction of malicious Web sites while maintaining a very low false positive rate.

We begin the remainder of this chapter with an explanation of the formal basis for our approach.

3.1 Approach

In this section, we provide a detailed discussion of our approach to classifying site reputation. We begin with an overview of the classification problem, followed by a discussion of the features we extract, and finally the set of machine learning classifiers we use for the study.

For our purposes, we treat URL reputation as a binary classification problem where positive examples are malicious URLs and negative examples are benign URLs. This learning-based approach to the problem can succeed if the distribution of feature values for malicious examples is different from benign examples, the ground truth labels for the URLs are correct, and the training set shares the same feature distribution as the testing set. (We revisit the last assumption by considering non-stationary distributions in Chapter 4.)

Significantly, we classify sites based only on the relationship between URLs and the lexical and host-based features that characterize them, and we do not consider two other kinds of potentially useful sources of information for features: the URL's page content, and the context of the URL (e.g., the page or email in which the URL is embedded). Although this information has the potential to improve classification accuracy, we exclude it for a variety of reasons. First, avoiding downloading page content is strictly safer for users. Second, classifying a URL with a trained model is a lightweight operation compared to first downloading the page and then using its contents for classification. Third, focusing on URL features makes the classifier applicable to any context in which URLs are found (Web pages, email, chat, calendars, games, etc.), rather than dependent on a particular application setting. Finally, reliably obtaining the malicious version

of a page for both training and testing can become a difficult practical issue. Malicious sites have demonstrated the ability to "cloak" the content of their Web pages, i.e., serving different content to different clients [NWC+07]. For example, a malicious server may send benign versions of a page to honeypot IP addresses that belong to security practitioners, but send malicious versions to other clients. Nonetheless, we show in Section 3.3 that classifying on lexical features of the URL and features about the host are sufficient for highly accurate prediction. (We propose investigating content-based features as future work in Section 7.2.1.)

3.1.1 Features

We categorize the features that we gather for URLs as being either lexical or host-based.

Lexical features: The justification for using lexical features is that URLs to malicious sites tend to "look different" in the eyes of the users who see them. Hence, including lexical features allows us to methodically capture this property for classification purposes, and perhaps infer patterns in malicious URLs that we would otherwise miss through ad-hoc inspection.

For the purpose of this discussion, we want to distinguish the two parts of a URL: the hostname and the path. As an example, with the URL `www.geocities.com/usr/index.html`, the hostname portion is `www.geocities.com` and the path portion is `usr/index.html`.

Lexical features are the textual properties of the URL itself (not the content of the page it references). We use a combination of features suggested by the studies of McGrath and Gupta [MG08] and Kolari et al. [KFJ06]. These properties include the length of the hostname, the length of the entire URL, as well as the number of dots in the URL — all of these are real-valued features. Additionally, we create a binary feature for each token in the hostname (delimited by '.') and in the path URL (strings delimited by '/', '?', '.', '=', '-' and '_'). This manner of allocating features is known as a "bag-of-words." Although we do not preserve the order of the tokens, we do make a distinction between tokens belonging to the hostname, the path, the top-level domain (TLD) and primary domain name (the

domain name given to a registrar). More sophisticated techniques for modeling lexical features are available, such as Markov models of text. However, even with the bag-of-words representation, we can achieve very accurate classification results (see Section 3.3.2).

Host-based features: Host-based features describe "where" malicious sites are hosted, "who" they are managed by, and "how" they are administered. The reason for using these features is that malicious Web sites may be hosted in less reputable hosting centers, on machines that are not conventional Web hosts, or through disreputable registrars.

The following are properties of the hosts (there could be multiple) that are identified by the hostname part of the URL. We note some of these features overlap with lexical properties of the URL.

1. *IP address properties* — Are the IPs of the A, MX or NS records in the same autonomous systems (ASes) or prefixes as one another? To what ASes or prefixes do they belong? If the hosting infrastructure surrounding malicious URLs tends to reside in a specific IP prefix or AS belonging to an Internet service provider (ISP), then we want to account for that disreputable ISP during classification.

2. *WHOIS properties* — What is the date of registration, update, and expiration? Who is the registrar? Who is the registrant? Is the WHOIS entry locked? If a set of malicious domains are registered by the same individual, we would like to treat such ownership as a malicious feature. Moreover, if malicious sites are taken down frequently, we would expect the registration dates to be newer than for legitimate sites.

3. *Domain name properties* — What is the time-to-live (TTL) value for the DNS records associated with the hostname? Additionally, the following domain name properties are used in the SpamAssassin Botnet plugin for detecting links to malicious sites in emails: Does the hostname contain "client" or "server" keywords? Is the IP address in the hostname? Is there a PTR record for the host? Does the PTR record in-turn resolve one of the host's

IP addresses (known as having a "full-circle" record)?

4. *Blacklist membership* — Is the IP address in a blacklist? For the evaluations in Section 4.5, 55% of malicious URLs were present in blacklists. Thus, although this feature is useful, it is still not comprehensive.

5. *Geographic properties* — In which continent/country/city does the IP address belong? As with IP address properties, hotbeds of malicious activity could be concentrated in specific geographic regions.

6. *Connection speed* — What is the speed of the uplink connection (e.g., broadband, dial-up)? If some malicious sites tend to reside on compromised residential machines (connected via cable or DSL), then we want to record the host connection speed.

This list of features we use is not exhaustive, as there is always the possibility of generating or aggregating new meta-information about the URL such as popularity rankings in Netcraft, indexing in Google, etc. Nevertheless, the list is still extensive (e.g., 17 feature categories yield 30,000–50,000 features in Section 3.2), as it represents many pieces of information about the URL and host (much of which is publicly available) that we can collect efficiently through the automated crawling tools at our disposal.

3.1.2 Classification Models

We use the features described in the previous section to encode individual URLs as very high dimensional feature vectors. Most of the features are generated by the "bag-of-words" representation of the URL, registrar name, and registrant name; binary features are also used to encode all possible ASes, prefixes and geographic locales of an IP address. The resulting URL descriptors typically have tens of thousands of binary features.

The high dimensionality of these feature vectors poses certain challenges for classification. Though only a subset of the generated features may correlate with malicious Web sites, we do not know in advance which features are relevant.

More generally, when there are more features than labeled examples, we enter the regime in which statistical models are most prone to overfitting.

In this section, we briefly review the models that we studied for classification, as well as their relative strengths and weaknesses. For all these models, we adopt the following notation. We use n to denote the number of labeled examples in the training set and d to denote the dimensionality of the feature space (i.e., the number of features). We use $\mathbf{x} \in \Re^d$ to denote a feature vector and x_j to denote its jth component. Finally, we use $y \in \{0, 1\}$ to denote the label of an example, with $y=1$ for malicious sites and $y=0$ for benign ones.

Though individual classifiers differ in their details, the same basic protocol applies to all the models we consider here. The classifiers are trained on labeled examples to learn a decision rule that can ultimately be applied to unlabeled examples. Given an input $\mathbf{x}$, the trained classifiers return a real-valued output $h(\mathbf{x})$ that we threshold to obtain a binary prediction. We hold out a small subset of the training data to determine this threshold t, which can be chosen to reach a desired tradeoff between false positives and false negatives. Finally, for $h(\mathbf{x}) \geq t$, we classify the site as malicious, and for $h(\mathbf{x}) < t$, we classify the site as benign.

We provide further details on the individual classifiers below. In particular, for each classifier, we briefly describe its testing process (how it predicts a label from the features of URLs) and its training process (how it learns the decision rule to predict these labels).

Naive Bayes: Commonly used in spam filters, this basic model assumes that, for a given label, the individual features of URLs are distributed independently of the values of other features [Bis06]. Letting $P(\mathbf{x}|y)$ denote the conditional probability of the feature vector given its label, the model assumes $P(\mathbf{x}|y) = \prod_{j=1}^{d} P(x_j|y)$. Then, from Bayes rule, assuming that malicious and benign Web sites occur with equal probability, we compute the posterior probability that the feature vector $\mathbf{x}$ belongs to a malicious URL as:

$$P(y=1|\mathbf{x}) = \frac{P(\mathbf{x}|y=1)}{P(\mathbf{x}|y=1) + P(\mathbf{x}|y=0)}. \tag{3.1}$$

Finally, the right hand side of eq. (3.1) can be thresholded to predict a binary label for the feature vector $\mathbf{x}$.

The Naive Bayes classifier is most easily trained by computing the conditional probabilities $P(x_j|y)$ from their maximum likelihood estimates [Bis06]. For real-valued features, we model $P(x_j|y)$ by a Gaussian distribution whose mean and standard deviation are computed over the jth component of feature vectors in the training set with label y. For binary-valued features, we estimate $P(x_j=1|y)$ as the fraction of feature vectors in the training set with label y for which the jth component is one.

The model parameters in the Naive Bayes classifier are estimated to maximize the joint log-likelihood of URL features and labels, as opposed to the accuracy of classification. Optimizing the latter typically leads to more accurate classifiers, notwithstanding the increased risk of overfitting.

Support Vector Machine (SVM): SVMs are widely regarded as state-of-the-art models for binary classification of high dimensional data. SVMs are trained to maximize the margin of correct classification, and the resulting decision boundaries are robust to slight perturbations of the feature vectors, thereby providing a hedge against overfitting. The superior generalization abilities of SVMs have been borne out by both theoretical studies and experimental successes [SS02].

The decision rule in SVMs is expressed in terms of a kernel function $K(\mathbf{x}, \mathbf{x}')$ that computes the similarity between two feature vectors and non-negative coefficients $\{\alpha_i\}_{i=1}^n$ that indicate which training examples lie close to the decision boundary. SVMs classify new examples by computing their (signed) distance to the decision boundary. Up to a constant, this distance is given by:

$$h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i (2y_i - 1) K(\mathbf{x}_i, \mathbf{x}), \tag{3.2}$$

where the sum is over all training examples. The sign of this distance indicates the side of the decision boundary on which the example lies. In practice, the value of $h(\mathbf{x})$ is thresholded to predict a binary label for the feature vector $\mathbf{x}$.

SVMs are trained by first specifying a kernel function $K(\mathbf{x}, \mathbf{x}')$ and then computing the coefficients α_i that maximize the margin of correct classification on the training set. The required optimization can be formulated as an instance of quadratic programming, a problem for which many efficient solvers have been

developed [CL]. In our study, we experimented with both linear and radial basis function (RBF) kernels.

Logistic Regression: This is a simple parametric model for binary classification where examples are classified based on their distance from a hyperplane decision boundary [HTF01]. The decision rule is expressed in terms of the sigmoid function $\sigma(z) = [1 + e^{-z}]^{-1}$, which converts these distances into probabilities that feature vectors have positive or negative labels. The conditional probability that feature vector $\mathbf{x}$ has a positive label $y=1$ is the following:

$$P(y=1|\mathbf{x}) \;=\; \sigma(\mathbf{w}\cdot\mathbf{x} + b), \tag{3.3}$$

where the weight vector $\mathbf{w} \in \Re^d$ and scalar bias b are parameters to be estimated from training data. In practice, the right hand side of eq. (3.3) is thresholded to obtain a binary prediction for the label of the feature vector $\mathbf{x}$.

We trained models for logistic regression using a regularized form of maximum likelihood estimation. Specifically, we chose the weight vector $\mathbf{w}$ and bias b to maximize the objective function:

$$\mathcal{L}(\mathbf{w}, b) = \sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i) - \gamma \sum_{\alpha=1}^{d} |w_\alpha|. \tag{3.4}$$

The first term computes the conditional log-likelihood that the model correctly labels all the examples in the training set. The second term in eq. (3.4) penalizes large magnitude values of the elements in the weight vector $\mathbf{w}$. Known as ℓ_1-norm regularization, this penalty not only serves as a measure against overfitting, but also encourages sparse solutions in which many elements of the weight vector are *exactly* zero. Such solutions emerge naturally in domains where the feature vectors contain a large number of irrelevant features. The relative weight of the second term in eq. (3.4) is determined by the regularization parameter. We determined the value of γ in our experiments by cross validation.

We included ℓ_1-regularized logistic regression for its potential advantages over Naive Bayes and SVMs in our particular domain. Unlike Naive Bayes classification, the parameters in logistic regression are estimated by optimizing an objective function that closely tracks the error rate. Unlike SVMs, ℓ_1-regularized

logistic regression is especially well suited to domains with large numbers of irrelevant features. (In large numbers, such features can drown out the similarities between related examples that SVMs expect to be measured by the kernel function.) Finally, because ℓ_1-regularized logistic regression encourages sparse solutions, the resulting models often have decision rules that are easier to interpret in terms of relevant and irrelevant features.

3.2 Data Sets

This section describes the data sets that we use for our evaluation. For benign URLs, we used two data sources. One is the DMOZ Open Directory Project [Net]. DMOZ is a directory whose entries are vetted manually by editors. The editors themselves go through a vetting process, with editors given responsibility for larger portions of the directory as they gain trust and experience. The second source of benign URLs was the random URL selector for Yahoo's directory. Visiting `http://random.yahoo.com/bin/ryl` multiple times can generate a sample of this directory.

We also drew from two sources for URLs to malicious sites: PhishTank [Ope] and Spamscatter [AFSV07]. PhishTank is a blacklist of phishing URLs consisting of manually-verified user contributions. Spamscatter is a spam collection infrastructure from which we extract URLs from the bodies of those messages. Note that the malicious URL data sets have different scopes. PhishTank focuses on phishing URLs, while Spamscatter includes URLs for a wide range of scams advertised in email spam (phishing, pharmaceuticals, software, etc.). Both sources include URLs crafted to evade automated filters, while phishing URLs in particular may be crafted to visually trick users as well.

Table 3.1 summarizes the number and types of features in the data sets that we use in our evaluations. The four data sets consist of pairing 15,000 URLs from a benign source (either Yahoo or DMOZ) with URLs from a malicious source (5,500 from PhishTank and 15,000 from Spamscatter). We refer to these sets as the DMOZ-PhishTank (DP), DMOZ-Spamscatter (DS), Yahoo-PhishTank (YP),

Table 3.1: Breakdown of feature types for data sets used in evaluations.

Feature type	**DP**	**YP**	**DS**	**YS**
DNS NS record	10818	4186	10838	3764
WHOIS info	9070	3959	8702	3525
Hostname	7928	3112	7797	2959
TLD + domain	5787	2186	5609	1980
DNS A record	5627	2470	5207	1862
Geographic	4554	2250	4430	1836
Path tokens	4363	8021	8390	11974
Last token of path	1568	3751	2892	5071
DNS MX record	1323	497	1422	559
TLD	319	284	102	51
Connection speed	31	29	30	30
DNS TTL	14	13	14	12
Blacklists	7	7	7	7
WHOIS dates	6	6	6	6
Spamassassin plugin	5	5	5	5
IP address misc.	4	4	4	4
Lexical misc.	3	3	3	3
All features	51427	30783	55458	33648

and Yahoo-Spamscatter (YS) sets. We collected URL features between August 22, 2008 – September 1, 2008.

We normalized the real-valued features in each feature set to lie between zero and one, shifting and rescaling each real-valued feature so that zero and one corresponded to the minimum and maximum values observed in the training set. Values outside this range in the testing set were clipped to zero or one as appropriate. The normalization served to equalize the range of the features in each feature set, both real-valued and binary. Intuitively, it reflects our prior belief that the real-valued and binary-valued features are equally informative and therefore should be calibrated on the same measurement scales.

One further complication arises due to undefined, or missing, features. Many real-valued features are undefined for large numbers of examples in the data set (e.g., DNS time-to-live values). We handled missing values using the following standard heuristic: for each real-valued feature, we defined an extra binary feature indicating whether the feature was defined. This heuristic enables the classifiers to

learn how to treat missing features from the way they appear in the training set.

3.3 Evaluation

In this section, we evaluate the effectiveness of the classifiers on identifying URLs to malicious sites. Specifically, we want to answer the following questions: Does using more features lead to more accurate classification? What is the most appropriate classification model to use? What is the impact on accuracy if we tune the classifier for lower false positives? Can we effectively classify data from one source with a model that was trained on a different data source? What trends do we see among the relevant features? And what do the misclassified examples have in common?

3.3.1 Methodology

We start by describing our experimental methodology. For each feature set and data set in our experiments, we perform classification over 10 random splits of the data, and the splits are 50/50 between training and testing. We learn a decision threshold t that will minimize the overall classification error (see Section 3.1.2 for methodology of learning t). We show the average classification results of those 10 splits.

We ran our experiments on a machine with 2 dual-core 2.33 GHz Xeon processors with 4 GB memory. Memory exhaustion was not an issue, but typical usage was on the order of a few hundred megabytes. We implemented a Naive Bayes solver in MATLAB. For the SVM solvers, we used the `.mex` implementations of LIBSVM [CL] and LIBLINEAR [FCH+08] that interface with MATLAB. We implemented a custom optimizer for ℓ_1-regularized logistic regression in MATLAB using multiplicative updates [SPS07].

3.3.2 Feature Comparison

Our first experiments on URL classification were designed to explore the potential benefits of considering large numbers of automatically generated features as opposed to small numbers of manually chosen features. Figure 3.1 compares the classification error rates on the four data sets described in Section 3.2 using nine different sets of features. The different feature sets involve combinations of features reflecting various practical considerations. For brevity, we only show detailed results from ℓ_1-regularized logistic regression (LR), which yields both low error rates (see Section 3.3.3) and also highly interpretable models (see Section 3.3.6). However, the other classifiers also produce qualitatively similar results.

Table 3.2 shows the total number of features in each feature set for the Yahoo-PhishTank data set. For these experiments, we also report the number of *relevant* features that received non-zero weight from ℓ_1-regularized logistic regression. (The other three data sets yield qualitatively similar results.) The feature sets are listed in ascending order of the total number of features. In what follows, we describe the detailed compositions of these feature sets, as well as their effects on classification accuracy.

We start with a "Basic" feature set that corresponds to the set of URL-related (not content-related) heuristics commonly chosen by various anti-phishing studies, including Fette et al. [FST07], Bergholz et al. [BCP+08] and CANTINA [ZHC07]. This set consists of four features: the number of dots in a URL, whether the hostname contains an IP address, the WHOIS registration date (a real-valued feature), and an indicator variable for whether the registration date is defined. Under this scheme, the classifier achieves a 10% error rate.

The next three feature sets (Botnet, Blacklist, and Blacklist + Botnet) represent the use of current spam-fighting techniques for predicting malicious URLs. The features for "Botnet" are from the SpamAssassin Botnet plugin (Section 3.1.1). These consist of five binary features indicating the presence of certain client- or server-specific keywords, whether the hostname contains an IP address, and two more features involving the PTR record of the host. With these features, we obtain a 15% error rate at best.

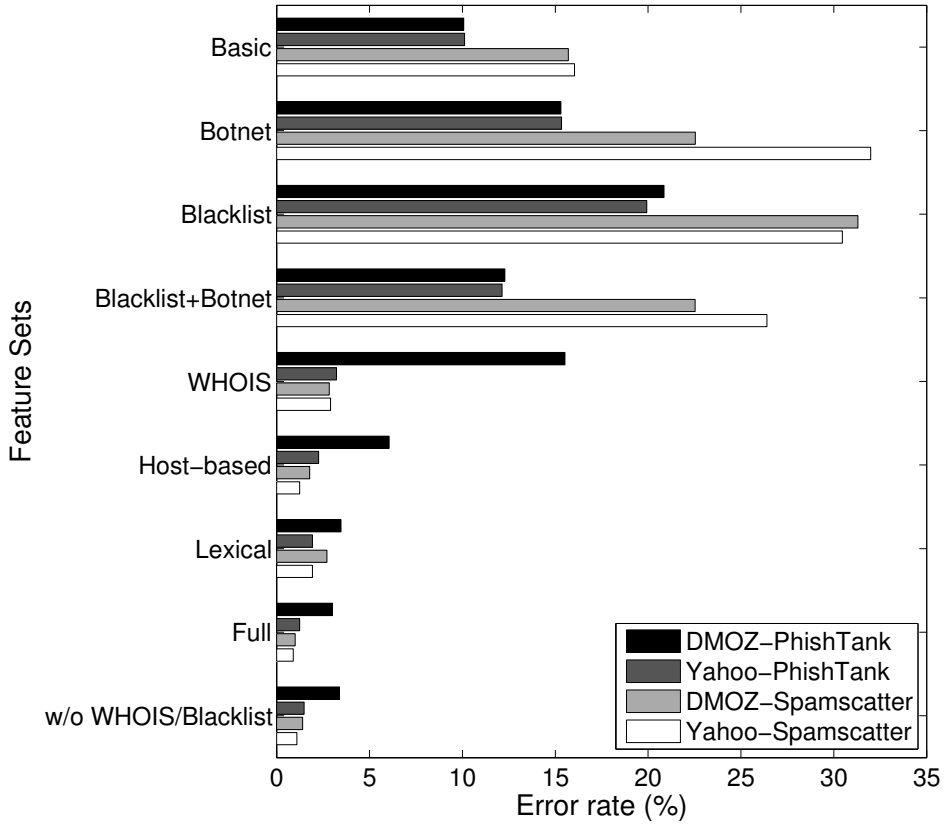


Figure 3.1: Error rates for LR with nine features sets on each of the four URL data sets. Overall, using more features improves classification accuracy.

The "Blacklist" feature set consists of binary variables for membership in six blacklists (and one white list) run by SORBS, URIBL, SURBL, and Spamhaus (discussed in Section 2.3.1). These lists combined provide at best a 20% error rate. When we combine the blacklists with the SpamAssassin Botnet plugin features in the "Blacklist+Botnet" feature set, the error rate improves to 12%.

These small features sets are reasonably effective, but including features that result in very large feature sets significantly improves classification accuracy.

The WHOIS registration date was a popular feature in previous studies [FST07][BCP+08][ZHC07]. Inspired by these examples, we also create a WHOIS feature set that includes the registration date as well as additional WHOIS properties. The "WHOIS" set we evaluated includes the registration, update, and expiration dates, as well as a bag-of-words representation of the registrar and reg-

istrant. The feature set grows to nearly 4,000 on the YP data set (Table 3.1.1), while reducing the error to 3–15% across the data sets. WHOIS features can provide useful information for differentiating malicious and benign URLs, since creators of legitimate sites are likely to adopt different registration patterns than creators of malicious sites (whose sites are taken down more frequently, or who want to anonymize their registration information).

WHOIS features, however, are just a subset of the available host-based features. If we also include the remainder of our host-based features — IP, DNS, and geographic properties as well as membership in a blacklist and SpamAssassin Botnet plugin features — the feature sets from our data sets grow to over 10,000 features. And these additional features are beneficial: the LR classifier has an even lower error rate of 1.2–6% across data sets. Using host-based features provides a richer set of properties for the classifier to capture phenomena such as malicious sites being hosted from a particular IP prefix, ISP, country and the like.

In addition to host-based features, we can also use lexical properties of the URLs themselves as potentially strong features (again, we are not considering the Web page content). That is, if the URLs themselves "look" malicious, then a lexical feature set can help us automatically learn the tell-tale strings that characterize a malicious or benign URL. For example, malicious sites may have legitimate-looking tokens appended to the sub-domain of the hostname, or have them embedded in the path of the URL for obfuscation purposes. In the "Lexical" set, we consider a "bag-of-words" representation of the tokens in the URL augmented with additional properties such as the length of the hostname, the length of the entire URL and the number of dots. This feature set is indeed effective, with a resulting error between 1.9% and 3.5% across data sets.

Combining the host-based and lexical features into the "Full" feature set, we obtain the lowest classification error among all feature sets at 0.9–3.0%. For YP, the FP/FN rates are 0.8%/2.6%.

Finally, blacklists and WHOIS information provide us known high-quality information for classifying URLs. But do the low classification error rates of the "Full" feature set critically depend on those particular features? The results for

Table 3.2: Total and relevant number of features for the LR classifier and the Yahoo-PhishTank data set. We reference the overall error rate from Figure 3.1.

	# Features		
Feature set	**Total**	**Relevant**	**Err%**
Basic	4	4	10.12
Botnet	5	5	15.34
Blacklist	7	6	19.92
Blacklist+Botnet	12	10	12.14
WHOIS	3967	727	3.22
Host-based	13386	1666	2.26
Lexical	17211	4488	1.93
Full	30597	3891	1.24
w/o WHOIS/Blacklist	26623	2178	1.48

"w/o WHOIS/Blacklist" show the error rates without WHOIS and blacklist features are 1.1–3.4%. Since the error rates across this feature set remain close to those of the "Full" set, we find that the remaining features provide sufficient information for achieving accurate classification results — using high-quality WHOIS and blacklist information is not necessary for excellent results. Thus, practitioners who are concerned about the query overhead of their implementations may choose to omit these features.

Having reviewed the composition and results for each feature set, we observe two noticeable trends in Figure 3.1: (1) Before "WHOIS," the DMOZ-PhishTank set has similar error rates to Yahoo-PhishTank, but starting with "WHOIS" the Yahoo-trained set has lower error. The shift occurs because the distribution of WHOIS and lexical features between DMOZ and Yahoo are distinct enough that certain URL tokens and certain WHOIS properties receive more weight in one data set than the other. (2) Prior to "WHOIS," the PhishTank-trained sets have better error rates because PhishTank Web sites have more undefined WHOIS registration dates (hence better "Basic" results), more IP addresses in the hostname (better "Botnet" results), and more URLs in blacklists (better "Blacklist" results). But starting with "WHOIS," the Spamscatter-trained sets do better because the introduction of registrar/registrant names provides a wealth of information for differentiating malicious and benign sites, and the distribution of lexical tokens in Spamscatter is distinct enough from benign sources to provide traction for accurate

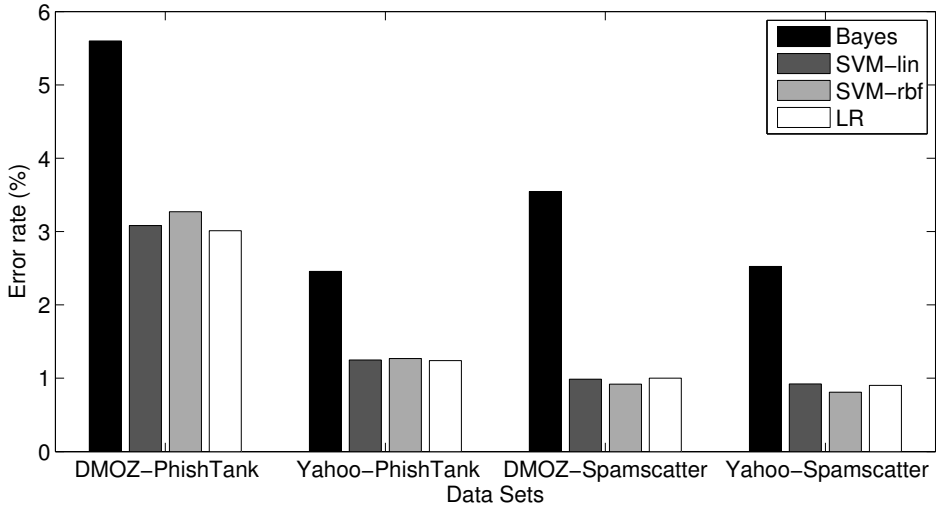


Figure 3.2: Overall error rates for the "Full" feature set using different classification algorithms and data sets.

classification results.

These results demonstrate that different data sets provide different feature distributions for distinguishing malicious and benign URLs. Rather than manually discovering and adjusting the decision rules for different data sets, machine learning techniques can adapt to these differing distributions by learning the appropriate decision rules automatically.

3.3.3 Classifier Comparison

Next we compare the accuracy of the four classifiers described in Section 3.1, each of which has different tradeoffs between model complexity and training execution time. We use the "Full" feature set because it yielded the best accuracy in Section 3.3.2.

Figure 3.2 compares the results of the classifiers: Naive Bayes (Bayes), SVM with a linear kernel (SVM-lin), SVM with an RBF kernel (SVM-rbf), and ℓ_1-regularized logistic regression (LR). The bars show the overall error rates for each classifier on each of our four data sets. Additionally, Table 3.3 shows the training and testing times for each classifier. The cross-validation time refers specifically to the cross-validation process used in LR to choose a regularization constant.

Although expensive, it can be effectively amortized when training over time.

Table 3.3: Training and testing times for the Yahoo-PhishTank data set.

	Bayes	**SVM-lin**	**SVM-rbf**	**LR**
Train	51 s	50 s	63 s	30 min
Test	87 s	91 s	170 s	90 s
Cross-validation	—	—	—	5 hrs

The SVM and LR classifiers have at least half of the error of Naive Bayes, which is not surprising given the models that we chose. Naive Bayes is a classifier that sees wide-spread use in spam filters and related security applications, in part because the training and testing performance of Naive Bayes is fast. However, the benefit of reduced training time is outweighed in this case by the benefit of using classifiers whose explicit goal is to minimize errors. This tradeoff is particularly worthwhile when we are dealing with a large feature set.

As mentioned in Section 3.1.2, SVM classifiers can perform poorly if irrelevant features in a large feature set make the kernel functions poor measures of similarity. Given that the difference in error between SVM and LR is so small, though, this problem did not materialize in our data set. As such, we continue to show the results for LR for the remaining experiments because of its interpretability, which is useful for understanding how the model performs (Section 3.3.6) and how it might be improved (Section 3.3.7).

3.3.4 Tuning False Positives & Negatives

An advantage of using these models is that they can trade off false positives and false negatives. We have seen in the previous sections that classifying with the "Full" feature set yields very low overall error rates. For policy reasons, however, we may not want to choose a decision threshold t to minimize the overall error rate. Instead, we may want to tune the threshold to have very low false positives at the expense of more false negatives (or vice versa). For instance, practitioners may be concerned that false positive classifications on legitimate sites result in lawsuits — in this case they would want to set the threshold higher to reduce the expected false positive rate.

Consider blacklisting as a motivating example. Blacklisting has the intrinsic advantage that it will have very low false positives because blacklisted sites go through a manual vetting process. Suppose a network administrator wants to take advantage of the benefits of classifying over a full feature set while maintaining the low false positives of a blacklist-only policy as applied to URL classification. To do so, we can select a threshold for the full feature set that will yield a false positive rate competitive with blacklisting while having a much lower false negative rate.

An ROC graph is a typical method for visualizing the sensitivity of classification models to the decision threshold t. In an ROC graph, the x-axis shows the false positive rate (fraction of benign URLs identified as malicious) and the y-axis shows the true positive rate (fraction of malicious URLs identified as malicious, calculated as 1− false negative rate). Each point on the ROC curve represents the true positive and false positive rate of that model for a particular decision threshold t. The curve for a perfect classifier would have points occupying the upper-left corner at $(x, y) = (0, 1)$, where there are no false negatives (i.e., 100% true positives) and no false positives.

Figure 3.3 shows the results of this experiment as an ROC graph with respect to the decision threshold t. We see that even if we tune the decision threshold of the full-featured classifier to the same false positives as a blacklist-only classifier, the full-featured classifier still predicts malicious URLs with much better accuracy than with blacklists alone.

3.3.5 Mismatched Data Sources

Coupling the appropriate classifier with a large feature set can yield a highly accurate classifier when trained and tested on disjoint sets of the same data source. But do these results hold when training and testing examples are drawn from different data sources? To answer this question, we experiment with training and testing on various combinations of benign and malicious sources of URLs. (We use the abbreviations defined in Section 3.2 to refer to each combination of data sources, e.g., YP for Yahoo-PhishTank.)

Table 3.4 shows classification results using ℓ_1-regularized logistic regression.

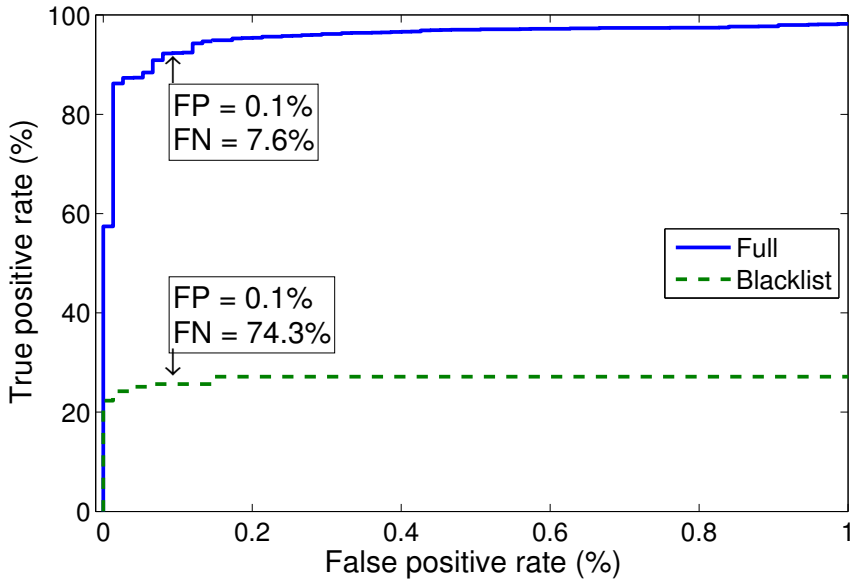


Figure 3.3: Tradeoff between false positives and false negatives: ROC graph for LR over an instance of the Yahoo-PhishTank data set using (1) the "Full" feature set and (2) blacklists only. We highlight the points where the false positives are tuned to 0.1%.

Each row represents a pairing of benign and malicious URL training sources, and each column represents a pairing of benign and malicious URL testing sources. As expected, the entries along the NW-SE diagonal yield the lowest classification errors because they represent matching similar data sources (these numbers are repeated from previous experiments). When only the benign URL source is mismatched (e.g., YS and DS), error increases due to higher false positives. And if only the malicious URL source is mismatched (e.g., YS and YP), error increases due to higher false negatives. Nevertheless, we note that training on the Spamscatter set — which includes URLs advertising a wide range of scam sites — generally performs better on a mismatched source (YS-YP at 6%, DS-DP at 14%) than the PhishTank set (YP-YS at 21%, DP-DS at 23%) — which focuses only on phishing sites (Section 3.2).

Finally, if the training and testing sources are completely mismatched (SW-NE diagonal), the error ranges between 18–44%. Although better than random, the disparity in accuracy emphasizes the judicious selection of training data. This selection is of particular importance for use in a deployment: the training data

Table 3.4: Overall error rates when training on one data source and testing on another (possibly different) data source using the LR classifier.

	Testing			
Training	YS	YP	DS	DP
YS	0.90%	5.66%	9.38%	17.73%
YP	21.33%	1.24%	44.02%	33.54%
DS	13.55%	31.57%	1.00%	13.70%
DP	22.95%	3.06%	22.92%	3.01%
All sources (YDSP)	0.95%	2.40%	1.43%	3.19%

should reflect the "testing" environment in which the system is used. To that end, if we use all four training sources, the generalization ability of the classifier is strong across all testing sources (last row). Thus, in a deployed system practitioners will want to pay special attention to collecting training data that is representative.

3.3.6 Model Analysis

Besides high classification accuracy, LR has the advantages of performing automatic feature selection as well as providing an interpretable linear model of the training data. In particular, because the output of a linear model depends on the weighted sum of the features, the sign and magnitude of the individual parameter vector coefficients can tell us how individual features contribute to a "malicious" prediction or a "benign" prediction. Generally, positive coefficients correspond with malicious features while negative coefficients correspond with benign features. [1] Additionally, the training phase for ℓ_1-regularized logistic regression yields a *sparse* parameter vector $\mathbf{w}$ where the number of non-zero coefficients is much smaller than the total number of features, making it easier for us to focus on a smaller number of relevant features (a zero coefficient means that the corresponding feature will not contribute to the prediction outcome). These properties simplify the analysis of trends in malicious and benign features.

In this section, we analyze the model that LR constructed for one of the

[1] This interpretation overlooks complicated feature correlations that cannot be learned by a linear classifier, such as XOR interactions. For example, if $a \otimes b = 1$ is a strong predictor for a malicious site (where a and b are binary features), then it is difficult to interpret the maliciousness of a by itself because of the dependence on b's value.

Table 3.5: Breakdown of features for LR for an instance of the Yahoo-PhishTank data set. We show total number of features in each feature type category, the number of relevant (non-zero) features, as well as the number of selected features with negative (benign) and positive (malicious) coefficients.

			Sign(coeff.)	
Feature type	**Total**	**Relevant**	**-**	**+**
Path tokens	8021	419	95	324
DNS NS record	4186	1021	468	553
WHOIS info	3959	633	284	349
Last token of path	3751	76	17	59
Hostname	3112	315	123	192
DNS A record	2470	407	173	234
Geographic	2250	692	329	363
TLD + domain	2186	237	97	140
DNS MX record	497	70	34	36
TLD	284	40	15	25
Connection speed	29	19	9	10
DNS TTL	13	9	6	3
Blacklists	7	7	1	6
WHOIS dates	6	6	1	5
Spamassassin plugin	5	5	3	2
IP address misc.	4	3	3	0
Lexical misc.	3	3	0	3
All feature types	30783	3962	1658	2304

ten random splits of the Yahoo-PhishTank data set, and discuss trends that we observe in the model and correlate them with real-world phenomena. This example has 3,962 active (non-zero) features out of a total of 30,783 features. There are 1,658 negative (benign) coefficients and 2,304 positive (malicious) coefficients in the weight vector. Table 3.5 shows a breakdown of the different types of features selected in this LR model. Overall, we find that the automatic feature selection of certain feature groups is specific to the data set at hand, while other feature types indicate broader trends in malicious sites.

Not surprisingly, the weights learned for the blacklist features are all non-zero because the presence of domain names and IPs in one of the six blacklists, or its absence in the whitelist, is an accurate indicator of maliciousness.

Additionally, the weights selected for the "SpamAssassin plugin" features match their purpose as an indicator. For example, the model learned negative

weights for having server words in the hostname, PTR records, and full-circle records. And it learned positive weights for having client words or an IP address in the hostname.

The miscellaneous IP address properties indicate whether the A record of a host shares the same prefix or AS as its MX or NS record. These features are considered benign on all counts, which makes sense because hosting the Web site and email server in the same data center reflects a more orthodox hosting configuration that a customer would purchase through legitimate means, as opposed to a hosting infrastructure acquired by compromising a scattered array of machines.

Next, the classifier selected 40 out of the 284 top-level TLD features. Generic TLDs like '.gov', '.edu', '.com', '.org', and country-level TLDs like '.ca' and '.se' are the top-6 benign features by weight. By contrast, '.info', '.kr', '.it', '.hu', and '.es' form the top-6 malicious features by weight, domains correlated with some malicious sites.

For hostname tokens that do not include the TLD, the model selected 315 out of 3,112 features. Some interesting examples of malicious hostname tokens include the appearance of 'com' in the name, resulting from URLs to malicious sites that prepend the tokens of legitimate hostnames (e.g., from banks) as an obfuscation measure. And while it is no surprise to see that the presence of 'www' is considered a benign (albeit spoofable) feature, 'members' receives the most negative weight because of its co-occurrence with benign sites hosted on 'members.aol.com', 'members.tripod.com' and other free hosting services.

Although the URL's path tokens contribute the most features (about 8,000), the model selects 419 relevant path tokens during training, 324 of which are malicious. Among the automatically selected malicious features are 'account', 'webscr', 'login', 'ebayisapi', 'signin', 'banking', and 'confirm' — for comparison, these tokens the model learned automatically are also 7 out of the 8 manually chosen path tokens that were considered tell-tale phishing features in the study by Garera et al. [GPCR07]. However, additional selected tokens include 'images', 'com', 'www', and 'paypal'. Tokens like these indicate attempts by the site authors to spoof legitimate sites by including domain names within the URL path ('www.example.com').

Even 'http:' is considered a malicious feature, aiding in the prediction of URLs that attempt to hide legitimate-looking URLs in the path component (the case of using a legitimate domain's redirection facilities was more rare). Because trends in malicious sites can change over time, it is important to learn path tokens automatically instead of manually.

Out of the 3,959 WHOIS information features (mainly tokens in the names of the registrar and registrant of the domain name), the classifier selected 633 as relevant. As for dates in the WHOIS record, missing any of the three WHOIS dates (registration, update, expiration) is considered a malicious feature. Moreover, having a recent registration or update date is considered malicious — this corresponds with the behavior of malicious sites having newer registration dates because they are taken down more frequently. Having a newer expiration date is the only benign WHOIS date feature.

For connection speed features, the model selected 19 out of 29. Having a T1 speed for the DNS A and MX records are the top-2 benign features, while a residential cable, DSL or satellite connection hosting an address in the A, MX or NS record of an entry is considered malicious. These features reflect the phenomenon of malicious sites hosted on compromised machines in residential ISPs.

Finally, the selected DNS A/MX/NS record features and geographic features correspond to hosting activity associated with various regions of the Internet address space. For example, IP ranges belonging to Google, Yahoo and AOL are treated as benign features. Also, having DNS NS records in IP ranges belonging to major registrars such as RIPE (the network information center for Europe) are considered benign features. However, having an NS record in one of the IP prefixes run by GoDaddy is considered a malicious feature — as discussed in Section 2.3.4, because of enforcement efforts that take down existing sites, criminals gravitate toward low-cost registrars when registering new sites.

Overall, the classifier was able to automatically select malicious and benign features for which domain experts had prior intuition. Perhaps more importantly, the classifier automatically selected new, non-obvious features that were highly predictive and yielded additional, substantial performance improvements.

When faced with such a reputation system, the ability of adversaries to evade detection depends on their ability to avoid conforming to the trends and selections determined by the classifier; we discuss the issue further in Section 3.4.

3.3.7 Error Analysis

A machine learning approach using a full feature set, consisting of both lexical and host-based features, yields a URL reputation classifier that has low false positive and low false negative rates. Despite the high accuracy of this approach, in this section we examine examples of misclassified URLs to reveal trends that can suggest refinements to our current approach, and also help us understand the limitations of using just URL features for classification (as opposed to additionally including features from Web page content, for example). For our analysis, we examine one of the instances of the Yahoo-PhishTank data sets from the LR classifier, chosen randomly, which had 60 false positives and 71 false negatives (other instances resulted in similar conclusions).

The most common cause of false positives (benign sites mistakenly classified as malicious) was due to sites hosted at disreputable ISPs. This is a case of guilt by association — our classifier penalizes legitimate sites hosted in the same AS or IP prefix as malicious sites. This set of false positives could be mitigated by examining the content of a site. Also, if reputation systems become more prevalent, such false positives imply that it would behoove legitimate site owners to seek service providers that have a reputation for cracking down on malicious sites hosted on their networks.

The false negatives (undetected malicious sites) fell into the following categories: (1) URLs to sites with benign tokens in the URL, (2) malicious sites using free hosting services (such as 'geocities.com' or 'tripod.com'), (3) compromised sites, (4) redirection services, (5) sites hosted in reputable geographic regions (such as the US), and (6) sites with international TLDs hosted in the US. The first category shows that if certain lexical features have substantial weights in classification, false negatives can result. Eliminating this false negative could be accomplished by more careful vetting of which URL tokens become features.

The last five categories are related because they reflect the situation where host-based features of a malicious URL *appear* to be non-malicious. However, the remedies for mitigating these kinds of false negatives differ. For example, eliminating malicious sites in category (3) is a matter of encouraging sites to maintain up-to-date patches of software, and category (5) reflects the practice of purchasing services from a reputable ISP; such remedies, though, are out of the control of a reputation system. However, the closely-related category (6) sites could be addressed by adding features that indicate whether the country of a site's TLD corresponds to the country where the ISP is located. Finally, mitigating false negatives in (2) and (4) may require examining the content of a site.

3.4 Evasion

If deployed and effective, adversaries will naturally try to invent methods for evading this statistical modeling approach. As we discussed in Section 3.1, the effectiveness of statistical modeling depends on a few key assumptions. When hosting malicious sites, adversaries can try to violate these assumptions to evade detection.

One approach for evasion is to reduce the information content in the lexical features of URLs. For example, using redirection services such as TinyURL produces fewer distinguishing lexical features (e.g., an identical host name coupled with a random token). Another approach is to acquire many benign host-based features, such as hiding behind the well-provisioned infrastructure provided by legitimate free hosting services (e.g., 'geocities.com' or 'tripod.com'). These aliasing techniques could provide adversaries methods for misclassifying their sites as benign.

As discussed in Section 3.3.6, the weights assigned to features in our classifiers reflect trends in the features of malicious and benign URLs. An adversary evading detection can strive to craft a site that can spoof sufficient benign features with high weights, and avoid acquiring sufficient malicious features, to appear as a benign site to the detection algorithm. For instance, '.org' is a benign TLD

according to our classifier, yet costs only $10/year to register.

There are, however, functional and cost-related limits to an adversary's ability to forge features. For instance, older WHOIS registration dates have high weights as benign features, but are difficult to casually obtain for malicious sites. Further, whether a site appears on a blacklist is out of its control. Whether such features are sufficient in the long term for differentiating benign from malicious sites remains an open question. As with any security approach facing an adaptable adversary, though, we envision having to evolve a deployment in response over time.

3.5 Summary

We have described an approach for classifying URLs automatically as either malicious or benign based on supervised learning across both lexical and host-based features. We argue that this approach is complementary to both blacklisting — which cannot predict the status of previously unseen URLs — and systems based on evaluating site content and behavior — which require visiting potentially dangerous sites. Further, we show that with appropriate classifiers it is feasible to automatically sift through comprehensive feature sets (i.e., without requiring domain expertise) and identify the most predictive features for classification.

However, an open issue is how to scale our approach to handle millions of URLs whose features evolve over time. We address the issue in the following chapter.

Chapter 3, in part, is a reprint of the material as it appears in Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) 2009. Ma, Justin; Saul, Lawrence K.; Savage, Stefan; Voelker, Geoffrey M. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Large-Scale Online Learning

In the previous chapter, we demonstrated the feasibility of using machine learning techniques over lexical and host-based features for predicting malicious URLs. That work is among several previous systems for URL classification that have relied on batch learning algorithms. However, once we enter a regime where large-scale training sets are at our disposal, the resource limitations imposed by batch methods become apparent. This chapter addresses two major issues: How can we use online learning to scale our approach to large-scale data (on the order of 10^6 examples and features)? And which online algorithms are best suited to the task?

We argue that online methods are far better suited to the practical nature of this problem for two reasons: (1) online methods can process large numbers of examples far more efficiently than batch methods; (2) we need to adapt to changes in malicious URLs and their features over time.

To demonstrate this approach, we have built a URL classification system that uses a live feed of labeled URLs from a large Web mail provider, and that collects features for the URLs in *real time* (see Figure 4.2). Using this data, we show that online algorithms can be more accurate than batch algorithms in practice because the amount of data batch algorithms can train on is resource-limited. We compare classical and modern online learning algorithms and find that modern algorithms can achieve accuracies up to 99% over a balanced data set. Finally, we show that continuous retraining over newly-encountered features is critical for

Table 4.1: Feature breakdown on Day 100 of the experiments.

Lexical		Host-Based	
Feature type	**Count**	**Feature type**	**Count**
Hostname	835,764	WHOIS info	917,776
Primary domain	738,201	IP prefix	131,930
Path tokens	124,401	AS number	39,843
Last path token	92,367	Geographic	28,263
TLD	522	Conn. speed	52
Lexical misc.	6	Host misc.	37
Lexical	1,791,261	Host-Based	1,117,901

adapting the classifier to detect new, malicious URLs.

We begin the rest of the chapter by motivating the need for large-scale detection and describing the online algorithms we use for classification. Next, we describe our data collection methodology and evaluate the models over our data set of labeled URLs. Finally, we conclude with an overall discussion.

4.1 Large-Scale Detection

Our goal is to detect malicious Web sites from the lexical and host-based features of their URLs. This section describes the scale of the application and motivates the need for online learning.

We analyze lexical and host-based features described in Section 3.1.1 because they contain information about the URL and host that is straightforward to collect using automated crawling tools. Thus, the list of features is extensive, but not necessarily exhaustive.

For our data set (which we describe in more detail in Section 4.3), Table 4.1 lists the lexical and host-based feature types we consider and the number contributed by each type. Although lexical types account for the majority of features (due to bag-of-words representation), host-based features still comprise a sizable fraction — on Day 100, lexical types account for 62% of features and host-based types account for 38%.

Figure 4.1 shows the cumulative number of features for each day of the eval-

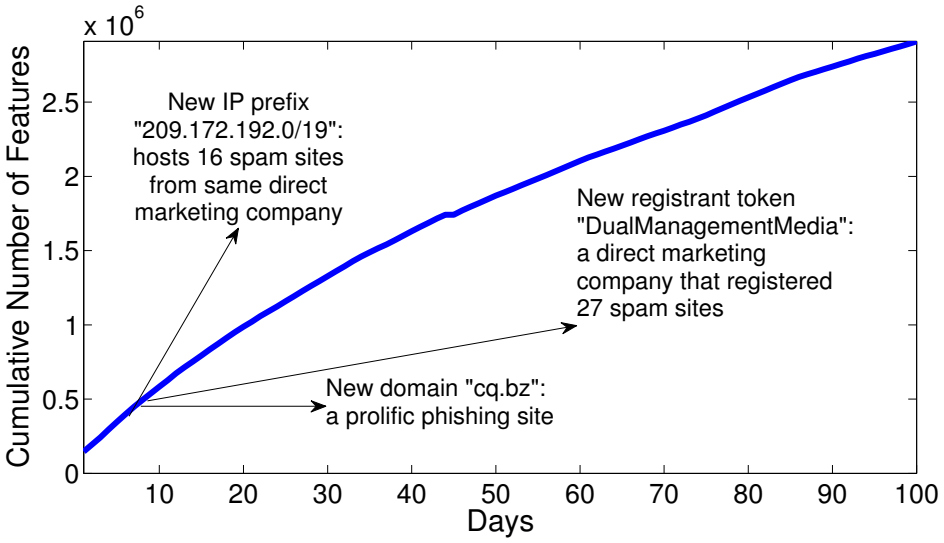


Figure 4.1: Cumulative number of features observed over time for our live URL feeds. We highlight a few examples of new features at the time they were introduced by new malicious URLs.

uations. Each day's total includes new features introduced that day and all old features from previous days (see Section 4.5 on our methodology for new features). For select days, we annotate the figure with particular features that appear for the first time in the data set. On Day 8, we show the company "DualManagementMedia" was responsible for purchasing 27 different spam domains. Around the same time, we observe a disreputable ISP (represented by IP prefix 209.172.192.0/19) for the first time. On that day, this ISP was responsible for hosting 16 different illegitimate sites.

The dimensionality grows quickly because we assign a binary feature for every token we encounter among the URL lexical tokens, as well as WHOIS and location properties. As we will show in Section 4.5.3, accounting for new features like the ones in Figure 4.1 is beneficial for detecting new malicious URLs.

4.2 Online Algorithms

This section briefly describes the online learning algorithms we use for our evaluations. Formally, the algorithms are trying to solve an online classification problem over a sequence of pairs $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_T, y_T)\}$, where each $\mathbf{x}_t$ is an example's feature vector and $y_t \in \{-1, +1\}$ is its label (in the online framework, this notation is more convenient). At each time step t during training, the algorithm makes a label prediction $h_t(\mathbf{x}_t)$, which for linear classifiers is $h_t(\mathbf{x}) = \text{sign}(\mathbf{w}_t \cdot \mathbf{x})$.

After making a prediction, the algorithm receives the actual label y_t. (If $h_t(\mathbf{x}_t) \neq y_t$, we record an error for time t.) Then, the algorithm constructs the hypothesis for the next time step h_{t+1} using h_t, $\mathbf{x}_t$ and y_t.

At the beginning, we did not have an *a priori* preference for particular online algorithms. Thus, the online methods we evaluate are a mix of classical and recent algorithms. We present the models in order of increasing sophistication with respect to the objective functions and the treatment of classification margin (which we can also interpret as classification confidence).

Perceptron: This classical algorithm is a linear classifier that makes the following update to the weight vector whenever it makes a mistake [Ros58]:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t \tag{4.1}$$

The advantage of the perceptron is its simple update rule. However, because the update rate is fixed, the perceptron cannot account for the severity of the misclassification. As a result, the algorithm can overcompensate for mistakes in some cases and undercompensate for mistakes in others.

Logistic Regression with Stochastic Gradient Descent: Many batch algorithms use gradient descent to optimize an objective function that is expressed as a sum of the examples' individual objective functions. Stochastic gradient descent (SGD) provides an *online* means for approximating the gradient of the original objective, whereby the model parameters are updated incrementally by the gradients of individual objectives. In this chapter we evaluate SGD as applied to logistic regression.

Let $P(y_t = +1|\mathbf{x}_t) = \sigma(\mathbf{w} \cdot \mathbf{x}_t)$ be the likelihood that example t's label is $+1$, where the sigmoid function is $\sigma(z) = [1 + e^{-z}]^{-1}$. Moreover, let $L_t(\mathbf{w}) = \log \sigma(y_t(\mathbf{w} \cdot \mathbf{x}_t))$ be the log-likelihood for example t. Then the update for each example in logistic regression with SGD is as follows:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \gamma \frac{\partial L_t}{\partial \mathbf{w}} = \mathbf{w}_t + \gamma \Delta_t \mathbf{x}_t \tag{4.2}$$

where $\Delta_t = \frac{y_t+1}{2} - \sigma(\mathbf{w}_t \cdot \mathbf{x}_t)$ and γ is a constant training rate. We do not decrease γ over time so that the parameters can continually adapt to new URLs. The update resembles a perceptron, except with a learning rate that is proportional to Δ_t, the difference between the actual and predicted likelihood that the label is $+1$. This multiplier allows the model to be updated (perhaps by a small factor) even when there is no prediction mistake.

SGD has received renewed attention because of recent results on the convergence of SGD algorithms and the casting of classic algorithms as SGD approximations [Bot98, BL04]. For example, the perceptron can be viewed as an SGD minimization of the hinge-loss function $Loss(\mathbf{w}) = \sum_t \max\{0, -y_t(\mathbf{w} \cdot \mathbf{x}_t)\}$.

Passive-Aggressive (PA) Algorithm: The goal of the passive-aggressive algorithm is to change the model as little as possible to correct for any mistakes and low-confidence predictions it encounters [CDSSS06]. Specifically, with each example PA solves the following optimization:

$$\begin{aligned} \mathbf{w}_{t+1} \quad \leftarrow \quad & \underset{\mathbf{w}}{\operatorname{argmin}} \quad \tfrac{1}{2}\|\mathbf{w}_t - \mathbf{w}\|^2 \\ & \text{s.t.} \qquad y_t(\mathbf{w} \cdot \mathbf{x}_t) \geq 1 \end{aligned} \tag{4.3}$$

Updates occur when the inner product does not exceed a fixed confidence margin — i.e., $y_t(\mathbf{w}_t \cdot \mathbf{x}_t) < 1$. The closed-form update for all examples is as follows:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha_t y_t \mathbf{x}_t \tag{4.4}$$

where $\alpha_t = \max\{\frac{1-y_t(\mathbf{w}_t \cdot \mathbf{x}_t)}{\|\mathbf{x}_t\|^2}, 0\}$. (The details of the derivation are in Crammer et al. [CDSSS06].) The PA algorithm has been successful in practice because the updates explicitly incorporate the notion of classification confidence.

Confidence-Weighted (CW) Algorithm: We provide a brief overview of the algorithm for this chapter and a more extensive overview in Chapter 6. The

idea behind confidence-weighted classification is to maintain a different confidence measure for each feature so that less confident weights are updated more aggressively than more confident weights. The "Stdev" update rule for CW is similar in spirit to PA. However, instead of describing each feature with a single coefficient, CW describes per-feature confidence by modeling uncertainty in weight w_i with a Gaussian distribution $\mathcal{N}(\mu_i, \Sigma_i)$ [DCP08, CDP09]. Let us denote $\boldsymbol{\mu}$ as the vector of feature means, and $\boldsymbol{\Sigma}$ as the *diagonal* covariance matrix (i.e., the confidence) of the features. Then the decision rule becomes $h_t(\mathbf{x}) = \text{sign}(\boldsymbol{\mu}_t \cdot \mathbf{x})$ — which is the result of computing the average signed margin $\mathbf{w}_t \cdot \mathbf{x}$, where $\mathbf{w}_t$ is drawn from $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, and then taking the sign.

The CW update rule adjusts the model as little as possible so that $\mathbf{x}_t$ can be correctly classified with probability η. Specifically, CW minimizes the KL divergence between Gaussians subject to a confidence constraint at time t:

$$\begin{aligned}(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) \leftarrow \underset{\boldsymbol{\mu},\boldsymbol{\Sigma}}{\text{argmin}}\ & \mathrm{D}_{KL}(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \| \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)) \\ \text{s.t. } & y_i(\boldsymbol{\mu} \cdot \mathbf{x}_t) \geq \Phi^{-1}(\eta)\sqrt{\mathbf{x}_t^\top \boldsymbol{\Sigma} \mathbf{x}_t}\end{aligned} \tag{4.5}$$

where Φ is the cumulative distribution function of the standard normal distribution. This optimization yields the following closed-form update:

$$\begin{aligned}\boldsymbol{\mu}_{t+1} &\leftarrow \boldsymbol{\mu}_t + \alpha_t y_t \boldsymbol{\Sigma}_t \mathbf{x}_t \\ \boldsymbol{\Sigma}_{t+1}^{-1} &\leftarrow \boldsymbol{\Sigma}_t^{-1} + \alpha_t \phi u_t^{-\frac{1}{2}}\ \text{diag}^2(\mathbf{x}_t)\end{aligned} \tag{4.6}$$

where α_t, u_t and ϕ are defined in Crammer et al. [CDP09]. However, we can see that if the variance of a feature is large, the update to the feature mean will be more aggressive. As for performance, the run time of the update is linear in the number of non-zero features in $\mathbf{x}$.

Overall, because the CW algorithm makes a fine-grain distinction between each feature's weight confidence, CW can be especially well-suited to detecting malicious URLs since our data feed continually introduces a dynamic mix of new and recurring features.

Related Algorithms: We experimented with nonlinear classification using online kernel-based algorithms such as the Forgetron [DSSS08] and the Projectron [OKC08]. To make computation tractable, these algorithms budget (or at

least try to reduce) the size of the support set used for kernel calculations. Our preliminary evaluations revealed no improvement over linear classifiers.

4.3 Data Collection

This section describes our live sources of labeled URLs and the system we deploy to collect features in real time. Figure 4.2 illustrates our data collection architecture, which starts with two feeds of malicious and benign URLs.

We obtain examples of malicious URLs from a large Web mail provider, whose live, real-time feed supplies 6,000–7,500 examples of spam and phishing URLs per day. The malicious URLs are extracted from email messages that users manually label as spam, run through pre-filters to extract easily-detected false positives, and then verified manually as malicious.

We randomly draw our examples of benign URLs from Yahoo's directory listing. A random sample from this directory can be generated by visiting the link `http://random.yahoo.com/bin/ryl`.

Combined, we collect a total of 20,000 URLs per day from the two URL feeds, and the average ratio of benign-to-malicious URLs is 2-to-1. With this 2-to-1 ratio, we implicitly value the cost of a false positive at twice the cost of a false negative — practitioners may prefer to adjust this training ratio to express a different policy on the relative cost of false positives vs. false negatives. We ran our experiments for 100 days, collecting nearly 2 million URLs (there were feed outages during Days 35–40). However, the feeds only provide URLs, not the accompanying features.

Thus, we deploy a system to gather features in real-time. The real-time aspect is important because we want the values to reflect the features a URL had when it was first introduced to the feed (which ideally reflects values for when it was introduced to the wild). For every incoming URL, our feature collector immediately queries DNS, WHOIS, blacklist and geographic information servers, as well as processing IP address-related and lexical-related features.

Our live feed is notably different from data sets such as the `webspam` set

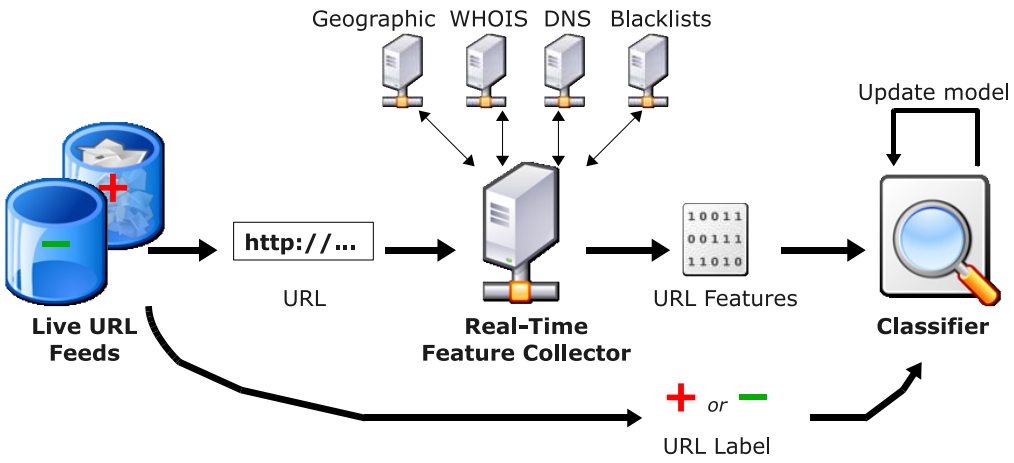


Figure 4.2: Overview of real-time URL feed, feature collection, and classification infrastructure.

from the PASCAL Large Scale Learning Challenge [SFYTS08]. Our application uses URLs as a starting point, and it is our responsibility to fetch lexical and host-based features in real-time to construct the data set on an ongoing basis. By contrast, the `webspam` set is a static representation of Web page content (not URLs) and individual pages do not have dates associated with them — i.e., `webspam` is not a temporal data set like the URL data set we collect.

Finally, our live feed provides a real-time snapshot of malicious URLs that reflect the evolving strategies of Internet criminals. The freshness of this data suggests that good classification results over the set will be a strong indicator of future success in real-world deployments.

4.4 Implementation

In this section we describe salient details regarding the implementation of our URL classification system.

At a high level, we implemented feature collection using a series of custom Bash and Ruby scripts. For a given URL, the individual components handling

lexical, IP address, WHOIS, DNS, blacklist and geographic features each output their results to an intermediate format. This intermediate format is then converted to a Matlab sparse matrix. After feature collection, we perform classification using online and batch algorithms, which we implement in Matlab.

The following sections describe more implementation details about the feature collection infrastructure and feature representation.

4.4.1 Feature Collection Infrastructure

We construct the feature vector for each URL in real time. When our feature collection server receives a URL, it attempts to query several external servers to construct the host-based portion of the feature vector. We explain the implementation details of host-based feature collection as follows:

- For IP address features, we look up the IP prefix and AS associated with a given IP address using a routing information base (RIB), which can be downloaded from the Route Views Project [Uni10]. There is a small, fixed overhead of loading and indexing the database (30–40 MB in size) in memory before feature collection begins. Once the database is loaded in memory, the lookup for IP prefixes and AS numbers is efficient. However, keeping the database up-to-date is an operational concern can practioners can address by periodically downloading the latest RIBs from Route Views.

- Parsing WHOIS data can be difficult because WHOIS entries are typically stored as flat text files with no standard format. We constructed a command-line PHP script around `PHPWhois` [JS10] to simplify parsing of WHOIS features. Nevertheless, WHOIS queries are high latency operations which take 1–3 seconds for most domain names and on the order of several seconds for domains hosted from smaller registrars. We set a query timeout of 7 seconds in our implementation to avoid undue delays in feature collection.

- We collect DNS data — e.g., IP addresses for the domain name's associated A, NS and MX records — by parsing the output of the `host` command.

Because DNS is structured to handle a high query volume, the latecy for collecting these features is low (less than a second).

- We query six blacklists (and one white list) run by SORBS, URIBL, SURBL and Spamhaus. This introduces seven binary features, and the query overhead is no more than performing a set of DNS queries.

- We collect geographic features using the NetAcuity service [Dig10]. Because we have a dedicated NetAcuity server for the campus, the query latency is very low.

4.4.2 Feature Representation

Because we implement the classification algorithms in Matlab, we have to represent the URL features explicitly as sparse vectors in a high-dimensional Euclidean space. Since new features are introduced continually, we must decide when to add new columns to the data matrix.

In our implementation, we collect data in day-by-day chunks and handle feature vector construction at the end of each day; Day N's feature vector becomes the union of the Day N's features with the previous days' features. Examples preceding the first occurrence of a newly added feature are assigned the value 0 for that feature. Moreover, the classification model's weight vector gets a 0 value for each newly added feature weight.

4.4.3 Suggestion for Production Deployment

In our evaluations, we deliberately decouple the act of feature collection from the act of classification because we want to create controlled evaluations. In particular, we perform data and feature collection *in real time*, but we can perform classification at a later time on data that retains its temporal characteristics. Nevertheless, constructing new feature vectors at the end of each day is too infrequent for a production deployment of our approach — we need a data representation that dynamically accounts for growing feature vectors.

Thus, for a production system we recommend storing a URL's features as a small hash table of <`feature, value`> pairs. The classification model's weight vector would likewise be a hash table. Because individual data vectors are sparse, efficient hash table-based implementations of the online algorithms are feasible.

4.5 Evaluation

In this section, we evaluate the effectiveness of online learning over the live URL feed. To demonstrate this effectiveness, we address the following questions: Do online algorithms provide any benefit over batch algorithms? Which online algorithms are most appropriate for our application? And is there a particular training regimen that fully realizes the potential of these online classifiers?

By "training regimen", we refer to (1) when the classifier is allowed to retrain itself after attempting to predict the label of an incoming URL, and (2) how many features the classifier uses during training.

For (1), we compare "continuous" vs. "interval-based" training. Under the "continuous" training regimen, the classifier may retrain its model after each incoming example (the typical operating mode of online algorithms). In the "interval-based" training regimen, the classifier may only retrain after a specified time interval has passed. In our experiments, we set the interval to be one day. Batch algorithms are restricted to interval-based training, since continuous retraining would be computationally impractical. Unless otherwise specified, we use continuous retraining for all experiments with online algorithms (and then evaluate the benefit of doing so in Section 4.5.3).

For (2), we compare training using a "variable" vs. "fixed" number of features. Under the fixed-feature regimen, we train using a pre-determined set of features for all evaluation days. For example, if we fix the features to those encountered up to Day 1, then we use those 150,000 features for the whole experiment (see Figure 4.1). Under the variable-feature regimen, we allow the dimensionality of our models to grow with the number of new features encountered; on Day 8, for instance, we classify with up to 500,000 features. Implicitly, examples that

were introduced before a feature i was first encountered will have value 0 for feature i. Unless otherwise specified, we use the variable-feature training regimen for all algorithms (and then evaluate the benefit of doing so in Section 4.5.3).

As for the sizes of the training sets, online algorithms implicitly train on a cumulative data set, since they can incrementally update models from the previous day. For batch algorithms, we vary the training set size to include day-long and multi-day sets (details in Section 4.5.1).

4.5.1 Advantages of Online Learning

We start by evaluating the benefit of using online over batch algorithms for our application in terms of classification accuracy — in particular, whether the benefit of efficient computation in online learning comes at the expense of accuracy. Specifically, we compare the online confidence-weighted (CW) algorithm against four different training set configurations of a support vector machine. For our canonical batch algorithm, we use the same `LIBLINEAR` [FCH+08] implementation of an SVM with a linear-kernel from Section 3.3.1. Evaluations with other batch algorithms such as logistic regression yielded similar results.

Figure 4.3 shows the classification rates for CW and for SVM using four types of training sets. We tuned all classifier parameters over one day of holdout data, setting $C = 100$ for SVM, and $\eta = 0.90$ for CW. The x-axis shows the number of days in the experiment, and the y-axis shows the cumulative error rate: the percentage of misclassified examples for all URLs encountered up to that date.

The SVM-once curve represents training once on Day 0's data and using that model for testing on all other days. The cumulative error steadily worsens to 3.5%, and the per-day false negative rate gets as high as 10–15%. These high error rates suggests that, to achieve better accuracy, the model must train on fresh data to account for new features of malicious *and* benign URLs encountered over time.

SVM-daily retrains only on data collected the previous day — e.g., Day 6 results reflect training on the URLs collected on Day 5, and testing on Day 6 URLs. The only exception is that we do not retrain during the feed outages on Days 35–40. As a result, the cumulative error is just under 3%, most of which is due to

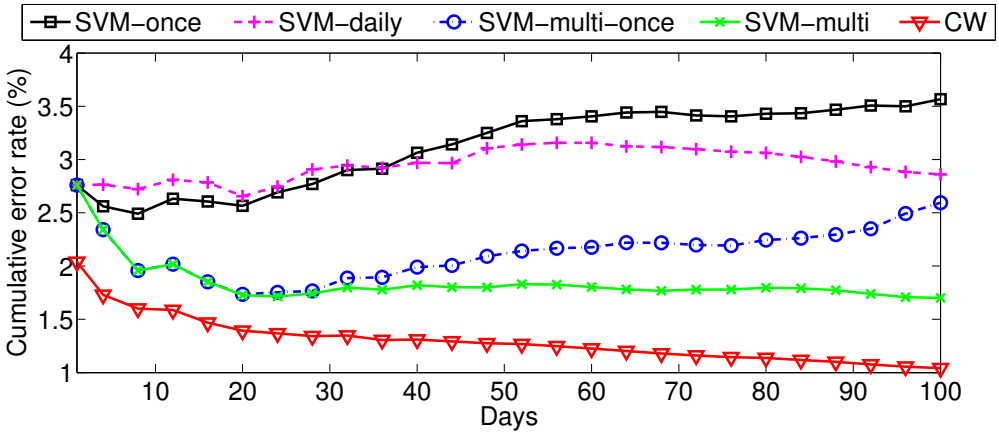


Figure 4.3: Cumulative error rates for CW and for batch algorithms under different training sets. Note the y-axis starts at 1%.

high per-day false negatives (5–15% on some days), whereas per-day false positives are around 1.5%. Although fresh data eventually helps SVM-daily improve over SVM-once, one day's training data is still insufficient.

We use multi-day training sets to address this issue by training on as much data as our evaluation machine with 4 GB RAM can handle (which is 14–17 days worth, or 280,000–340,000 examples). SVM-multi-once is the multi-day analogue to SVM-once. Here, SVM-multi-once trains on data from Days 0 to 16, and from Day 17 on it uses that fixed model for testing on subsequent days. The improvement over SVM-once shows the benefit of more training data, but the steadily worsening error again demonstrates the nonstationarity of the URL data set.

SVM-multi is the multi-day analogue of SVM-daily. Here, SVM-multi trains on the previous 14–17 days worth of data (depending on what can fit in memory). The resulting cumulative error reaches 1.8%. SVM-multi's improvement over SVM-multi-once suggests the URL feature distribution evolves over time, thus requiring us to use as much fresh data as possible to succeed. Overall, these results suggest that more training data yields better accuracy. However, this accuracy is fundamentally limited by the amount of computing resources available.

Fortunately, online algorithms do not have that limitation. Moreover, they have the added benefit that they can incrementally adapt to new data. As we see in

Figure 4.3, CW has higher accuracy than SVM-multi. Since the online algorithm is making a *single pass* over a cumulative training set, it does not incur the overhead of loading the entire data set in memory. Because its training is incremental, it is capable of adapting to new examples in real time, whereas batch algorithms are restricted to retraining at the next available interval (more on interval-based training in Section 4.5.3). These advantages allow the online classifier to have the best accuracy in our experiments.

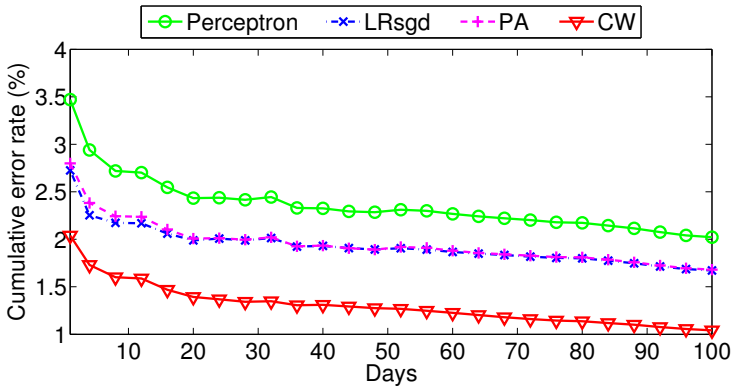
4.5.2 Comparison of Online Algorithms

Given the demonstrated benefits of online learning over batch learning, we next evaluate which of the online algorithms from Section 4.2 are best suited to malicious URL detection. The main issue that these experiments address is whether recent developments in online algorithms, which include optimizing different objective functions, adjusting for classification confidence, and treating features differently, can benefit the classifiers in our application.

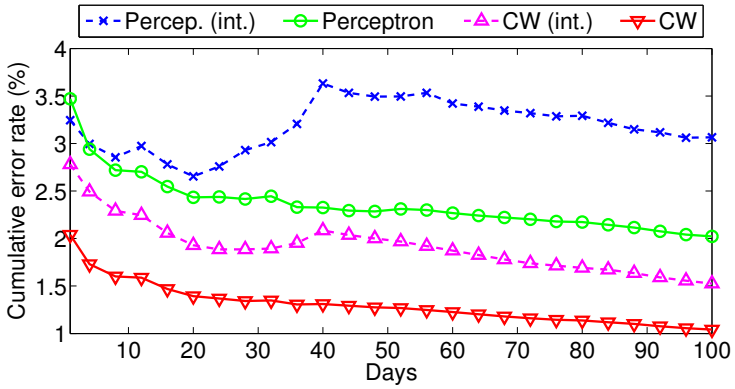
Figure 4.4(a) shows the cumulative error rates for the online algorithms. All algorithms in this experiment adopt the continuous training regimen. We also note that the error rates improve steadily over time for all classifiers, reaffirming that training on cumulative data is beneficial.

The perceptron is the simplest of the algorithms, but it also has the highest error rates across all of the days at around 2–3%. This result suggests that because the perceptron treats mistakes equally (and ignores all correct classifications), its updates are too coarse to accurately keep up with new examples. There needs to be a more fine-grain distinction between misclassified and correctly-classified examples with respect to their impact on model updates.

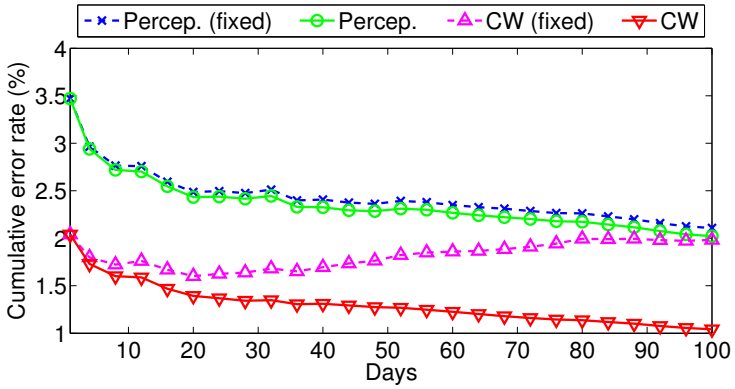
Both logistic regression with stochastic gradient descent (LRsgd) and the passive-aggressive (PA) algorithm achieve a cumulative error approaching 1.6%, improving over the perceptron results. (Here we tuned the LRsgd learning rate to $\gamma = 0.01$ over one day of holdout data.) Presumably, this improvement occurs because LRsgd and PA account for classification confidence. Specifically, LRsgd updates are proportional to Δ_t, and PA updates are proportional to the normalized



(a) Error rates for online algorithms. All use continuous/variable-feature training.



(b) Benefits of using continuous training over interval-based training.



(c) Benefits of using variable-feature sets over fixed-feature sets.

Figure 4.4: Comparing the effectiveness of various online algorithms, their use of continuous vs. interval training, and their use of fixed vs. variable feature sets.

classification margin α_t. These results are comparable to SVM-multi.

The CW results suggest that the final leap comes from *treating features differently* — both in terms of how they affect classification confidence, and how quickly they should be updated. With an error approaching 1%, CW clearly outperforms the other algorithms. Most of the gap between CW and the other online methods comes from CW's lower false negatives — CW has 1–2% false negatives per day, whereas others have 2–4%. We hypothesize the gap occurs because CW can update select portions of its model very aggressively to account for new malicious features, all without perturbing more established features.

Overall, we find that the more recent online algorithms outperform the simpler ones. Because the live combined URL feed contains a dynamic mix of new and recurring features, CW's per-feature confidence weighting can exploit that structure to achieve the best accuracy.

4.5.3 Training Regimen

In this section, we show that there is a significant advantage to continuous training vs. interval-based training. We also demonstrate that there is significant benefit to adding newly-encountered features as opposed to using a fixed feature set. The aforementioned training regimens can help online algorithms stay abreast of changing trends in URL features. Thus, choosing the right training regimen can be just as important as choosing the right algorithm.

Figure 4.4(b) shows the value of using continuous training over interval training with the CW and perceptron algorithms. The higher error rates for interval training show that there is enough variation between days that a model can become stale if it is not retrained soon enough. In particular, the higher number of false negatives for interval-trained CW is responsible for the persistent gap with continuously-trained CW. Notwithstanding the aforementioned feed outages on Days 35–40, the 1% error difference between continuous and interval-based perceptron is due to spikes in the false positive/negative rates for the interval-trained perceptron. (During the outage, interval-trained CW's performance does not degrade as severely because its per-feature learning rates ensure that low-variance

feature weights are not perturbed severely.) Thus, continuous retraining yields as much improvement for the simpler perceptron as it does for CW.

In addition to continuous retraining, accounting for new features is critical to an algorithm's success. Figure 4.4(c) shows the value of using variable-feature training over fixed-feature training. In this graph, "fixed features" means that we restrict the model to using the features encountered on Day 1 only (150,000 features total). We see that the performance for fixed-feature CW degrades to a point that it is no better than a perceptron. Interestingly, variable-feature perceptron only achieves a marginal improvement over fixed-feature perceptron. One explanation is that, even though variable-feature perceptron can occasionally benefit from adding new features, it does not update the new feature weights aggressively enough to correct for future errors. By contrast, the CW algorithm updates new features aggressively *by design*, and hence can reap the full benefits of variable-feature training.

Overall, continuous retraining with a variable feature set allows a model to successfully adapt to new data and new features on a sub-day granularity. And this adaptiveness is critical to the full benefits of online algorithms.

4.6 Proof-of-Concept

Given the promising experimental results from Section 4.5, we decided to implement a prototype of our URL classification system as a Web browser toolbar, which we call `GranolaBar`. In this proof-of-concept, our Firefox extension tells the user (1) the predicted "safety rating" of the currently loaded URL and (2) the predicted safety rating of a URL the user is thinking about visiting (i.e., when a user mouses over a link before clicking it).

As a user visits or mouses over a target URL, the browser sends the URL to our reputation server, which stores a linear classifier trained by confidence-weighted (CW) learning. The reputation server returns the result of the classification (expressed as a "safety percentage"), the top-10 highest-weight malicious features and top-10 highest-weight benign features back to the browser. The la-

tency of this query is typically 1–3 seconds because the reputation server gathers the URL's host-based features in real time (with WHOIS features dominating the query time).

Figure 4.5 demonstrates what the user sees when our plugin predicts the maliciousness of a real phishing site — before the user clicks on the phishing URL. The phishing site in question attempted to mimic an Authorize.Net login page. (Authorize.Net is a Web commerce service that helps merchants process credit card transactions, making it an appealing target for phishing attacks.) When we initially clicked the link on May 7, 2009, Firefox let us visit the site. However, when we tried to visit the phishing site again on May 12, 2009, Firefox's default blacklist protection issued a warning. Thus, our predictive classification approach identified the phishing site before it appeared on Firefox's blacklist.

Overall, this proof-of-concept is a promising demonstration that our approach can classify malicious sites before they appear on blacklists.

4.7 Summary

URL classification is a challenging task because the distribution of features that characterize malicious URLs evolves continually. With an eye toward ultimately constructing a real-time malicious URL detection system, we evaluated batch and online learning algorithms for our application to study their benefits and tradeoffs.

Experiments over a live URL feed revealed the limitations of batch algorithms in this setting, where we were forced to make a tradeoff between accuracy and coping with resource limitations (e.g., running out of memory). We demonstrated that recently-developed online algorithms such as CW can be highly accurate classifiers, capable of achieving classification accuracies up to 99%. Furthermore, we showed that retraining algorithms continuously with new features is crucial for adapting successfully to the ever-evolving stream of URLs and their features.

The success of confidence-weighted learning in detecting malicious URLs

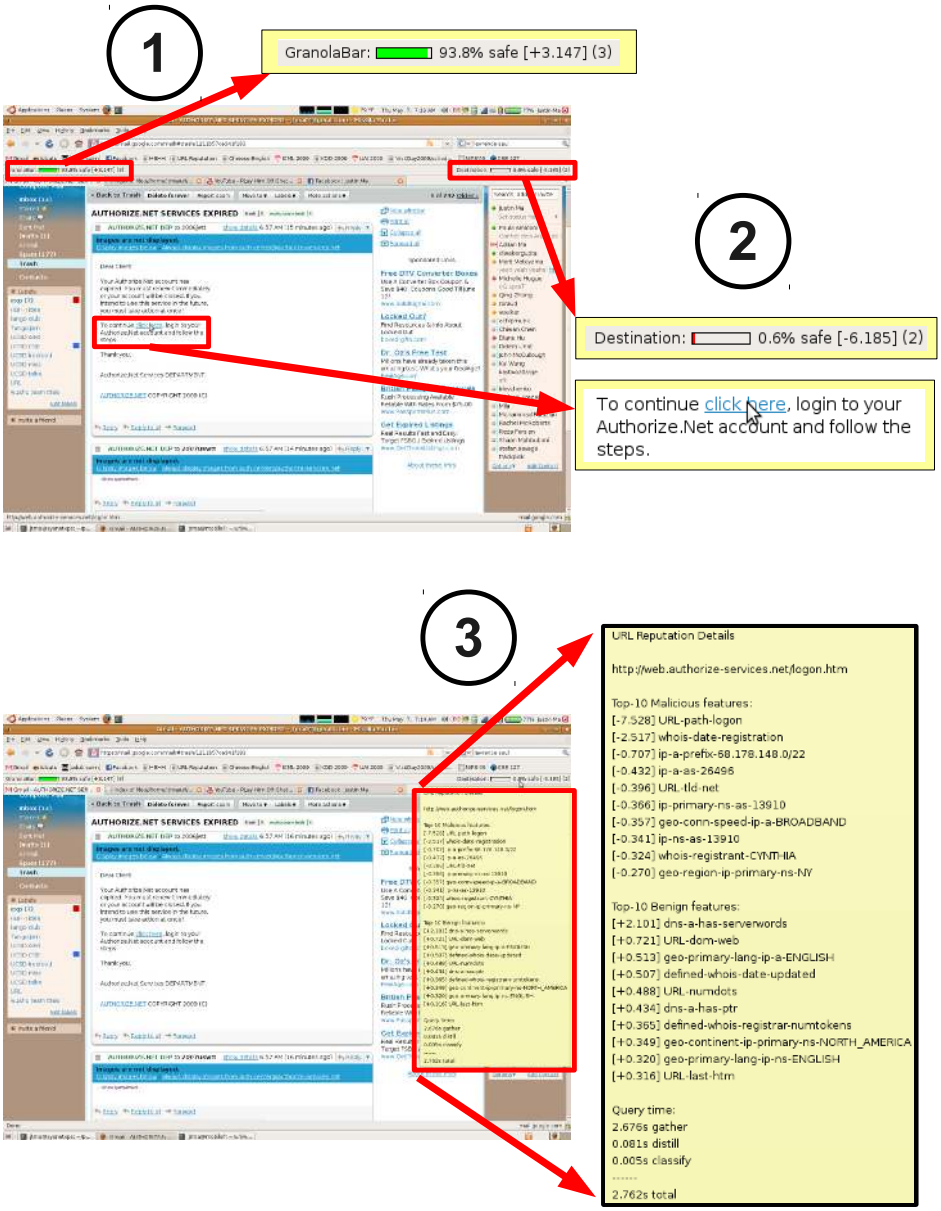


Figure 4.5: Proof-of-Concept Toolbar— (1) The toolbar rates the current Web mail page as safe. (2) Mousing over the phishing link in an email yields a warning that the moused-over URL is unsafe (the toolbar queries this URL's safeness rating in the background). (3) Mousing over the warning bar yields a pop-up window displaying the top malicious/benign features for the site, as well as query time.

illustrates the importance of accounting for feature-weight uncertainty in this and similar applications. However, CW is a recent development and is not the first online algorithm to incorporate model uncertainty. In the following chapter, we explore an older, related algorithm called Bayesian logistic regression and compare it against CW for malicious URL detection. In doing so, we gain more understanding about why CW performs well on this application.

Finally, we have made our data set available so that researchers interested in developing algorithms will have a large-scale, temporal data set at their disposal. The URL is `http://www.sysnet.ucsd.edu/projects/url/`.

Chapter 4, in part, is a reprint of the material as it appears in Proceedings of the International Conference on Machine Learning (ICML) 2009. Ma, Justin; Saul, Lawrence K.; Savage, Stefan; Voelker, Geoffrey M. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Uncertainty in Online Learning

In Chapter 4, we saw online algorithms that incorporate feature-weight uncertainty produced the best classification results for detecting malicious URLs. In this chapter, we develop a deeper understanding of these 2nd-order online algorithms (those that incorporate uncertainty) to determine why they work well for our application.

Issues of uncertainty arise whenever we attempt to predict the future from statistical models of the past. Typically, the past contains data that constrains and informs our future predictions, but does not provide enough information to identify the data's underlying model. In this chapter, we compare two approaches — one old, one new — for managing this uncertainty in statistical models of linear classification. We work in the online setting, where classifiers incrementally incorporate the evidence provided by each newly labeled example, then discard the example from memory after updating the decision boundary.

The first approach we study is Bayesian logistic regression (BLR). Bayesian methods are widely used for modeling uncertainty in parameter estimation [BS94]. For linear classification, this uncertainty is expressed as a probability distribution over the weight vector that defines the decision boundary. BLR assumes a prior distribution over this weight vector, then uses Bayes rule to compute the posterior distribution induced by labeled examples. New examples are classified by integrating the model predictions over this posterior distribution. Conceptually, BLR is naturally suited to online learning: Bayes rule dictates how to incorporate evidence

from successive labeled examples, yielding incremental updates for the posterior distribution over the weight vector. In general, however, exact implementations of BLR are impractical because the posterior distribution from Bayes rule cannot be computed in closed form. To circumvent this difficulty, implementations of BLR resort to some form of approximation. A common approximation is to model the posterior distribution over the weight vector by a multivariate Gaussian distribution [SL90, Mac92, JJ00]. Our implementations of BLR make this approximation, which in turn leads to simple and efficient update rules for online learning.

The second approach we study in this chapter is confidence-weighted (CW) learning [DCP08, CDP09], which we adopted in Chapter 4. Despite certain similarities with BLR in both concept and practice, CW grew out of a rather different tradition in statistical learning [CBCG05, CDSSS06, DSSS08, OKC08]. CW also models the uncertainty in the weight vector during online learning by a multivariate Gaussian distribution. But unlike BLR, this distribution is not sequentially updated by Bayes rule, but by solving a constrained optimization. This optimization ensures that with high probability, each newly labeled example is correctly classified by a large margin. For CW, the weight vector distribution is updated by the minimum amount required to satisfy this criterion.

In this chapter, we compare BLR and CW, highlighting their similarities as well as their fundamental differences. To our knowledge, we present the first parallel presentation of these algorithms that elucidates their common structure. This common structure is not immediately apparent from previous expositions. We also present the first experimental results comparing these two approaches. Most notably, in simple, controlled experiments on synthetic data, we show that BLR and CW can exhibit very different performance.

Beyond these experiments, we also compare BLR and CW on this dissertation's focus application of detecting malicious Web sites from suspicious URLs. We use the `URL Reputation` data set from Chapter 4, whose lexical and host-based (geographic, domain name, IP-based) features were gathered in real time and which contains nearly four months of data. A challenge of this application is that the feature set is constantly evolving; by Day 110, the URLs are represented by sparse

feature vectors with over two million elements. In Chapter 4, we found that CW outperformed many other approaches on this task; as independent practitioners seeking only the best results, we wondered if BLR — which models uncertainty in a different but time-honored fashion — would yield similar or perhaps further improvements.

The organization of this chapter is as follows. We begin by reviewing BLR and the approximations that have been developed for efficient, incremental updates to the posterior distribution. Next, we review CW learning and present a high-level comparison that highlights the similarities and differences with BLR. Finally, we present our experimental results and conclude with a discussion of our most significant findings.

5.1 Bayesian Logistic Regression

Given a weight vector $\mathbf{w}$, logistic regression models the binary response $y \in \{-1, +1\}$ to an input $\mathbf{x}$ as:

$$P(y|\mathbf{x}, \mathbf{w}) = \sigma(y\mathbf{x}^\top \mathbf{w}), \tag{5.1}$$

where $\sigma(z) = [1 + e^{-z}]^{-1}$ denotes the sigmoid function. In online learning, we attempt to estimate the weight vector $\mathbf{w}$ from a stream of labeled examples. Initially, after observing only a few examples, our estimate of $\mathbf{w}$ may be highly uncertain. However, as we observe more examples, our uncertainty in the decision boundary will evolve and in general decrease. The Bayesian approach to logistic regression provides a principled way to model this uncertainty and incorporate new evidence from successive labeled examples.

5.1.1 Bayesian Updating

More formally, let $D_{t-1} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{t-1}, y_{t-1})\}$ represent a sequence of $t-1$ labeled examples, and let the distribution $P(\mathbf{w}|D_{t-1})$ model our uncertainty in the weight vector $\mathbf{w}$ after observing these examples. Then to classify a new

example $\mathbf{x}_t$, we compute

$$P(y_t|D_{t-1}, \mathbf{x}_t) = \int d\mathbf{w} P(\mathbf{w}|D_{t-1}) P(y_t|\mathbf{x}_t, \mathbf{w}), \tag{5.2}$$

which integrates over our current uncertainty in the decision boundary characterized by $\mathbf{w}$. Likewise, to incorporate new evidence $(\mathbf{x}_t, y_t)$, we use Bayes rule:

$$P(\mathbf{w}|D_t) = \frac{P(\mathbf{w}|D_{t-1}) P(y_t|\mathbf{x}_t, \mathbf{w})}{\int d\mathbf{w} P(\mathbf{w}|D_{t-1}) P(y_t|\mathbf{x}_t, \mathbf{w})}, \tag{5.3}$$

which updates the posterior distribution over the weight vector from time $t-1$ to time t.

Though simple in concept, the above procedure is difficult to implement in practice. For our initial model of uncertainty in the weight vector $\mathbf{w}$, we assume a Gaussian prior distribution $P(\mathbf{w}|D_0)$ with mean $\boldsymbol{\mu_0}$ and covariance matrix $\boldsymbol{\Sigma}_0$. We also assume that $\boldsymbol{\Sigma}_0 = \omega \mathbf{I}$ is a scalar multiple of the identity matrix, with $\omega > 0$. But even with this simple choice, an exact evaluation of the high-dimensional, non-Gaussian integral in eqs. (5.2)–(5.3) is intractable.

As an alternative approach, we can approximate the non-Gaussian integral using Laplace or variational methods. We discuss these approximations in the following sections. We favor these approximations over sampling-based techniques [GRS96] because they lead to fast, closed-form updates.

5.1.2 Laplace Approximation

The Laplace approximation for Bayesian parameter estimation [Mac92] constructs a multivariate Gaussian distribution centered at the peak[1] of the posterior distribution $P(\mathbf{w}|D_t)$. Let us assume that $P(\mathbf{w}|D_{t-1})$ is a multivariate Gaussian at the previous time step. Then $P(\mathbf{w}, y_t|D_{t-1}, \mathbf{x}_t)$, which equals the numerator of eq. (5.3), is proportional to the following:

$$e^{\log \sigma(y_t x_t^\top \mathbf{w}) - \frac{1}{2}(\mathbf{w}-\boldsymbol{\mu}_{t-1})^\top \boldsymbol{\Sigma}_{t-1}^{-1} (\mathbf{w}-\boldsymbol{\mu}_{t-1})}. \tag{5.4}$$

Let $\mathbf{w}^*$ denote the mode of the posterior distribution: i.e., the value that maximizes eq. (5.4). As it turns out, we can express the mode as an additive correction

[1] See Spiegelhalter and Lauritzen [SL90] for a related approximation, which we do not pursue here.

to the posterior mean at the previous time step, where it takes the form $\mathbf{w}^* = \boldsymbol{\mu}_{t-1} + \alpha_t y_t \mathbf{s}_t$ (we define $\mathbf{s}_t = \boldsymbol{\Sigma}_{t-1}\mathbf{x}_t$). The scalar prefactor α_t can be computed by a one-dimensional Newton-Raphson procedure; see Appendix A for an efficient implementation of the Newton-Raphson method in this problem.

We construct a Gaussian approximation by taking the second-order Taylor expansion around the mode $\mathbf{w}^*$. As shorthand, let $z_t = y_t \mathbf{x}_t^\top \mathbf{w}^*$ and $\kappa_t = \sigma(z_t)\sigma(-z_t)$. Then the Laplace approximation can be written as:

$$P(\mathbf{w}, y_t | D_{t-1}, \mathbf{x}_t) \sim e^{-\frac{1}{2}(\mathbf{w}-\mathbf{w}^*)^\top \left[\boldsymbol{\Sigma}_{t-1}^{-1} + \kappa_t \mathbf{x}_t \mathbf{x}_t^\top\right](\mathbf{w}-\mathbf{w}^*)}. \tag{5.5}$$

We identify the right hand side as a multivariate Gaussian distribution with mean $\boldsymbol{\mu}_t = \mathbf{w}^*$ and inverse covariance matrix $\boldsymbol{\Sigma}_t^{-1}$ given by the bracketed term in the exponent. Using the matrix inversion lemma over $\boldsymbol{\Sigma}_{t-1}^{-1}$ and substituting for $\mathbf{w}^*$ yields the final update:

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \alpha_t y_t \mathbf{s}_t, \tag{5.6}$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t-1} - \beta_t \mathbf{s}_t \mathbf{s}_t^\top, \tag{5.7}$$

where as additional shorthand in eq. (5.7) we have defined $\beta_t = \kappa_t(1+\kappa_t \mathbf{x}_t^\top \mathbf{s}_t)^{-1}$. For prediction, we can use the Laplace method to compute $P(\hat{y}_t | D_{t-1}, \mathbf{x}_t) = \int d\mathbf{w}\, P(\mathbf{w}, \hat{y}_t | D_{t-1}, \mathbf{x}_t)$ approximately for outcomes $\hat{y}_t \in \{-1, +1\}$, then choose the label $\hat{y}_t$ that appears most likely under the approximation in eq. (5.5).

We show pseudocode for the Laplace approximation to BLR in option (i) of Algorithm 1. In very high dimensional feature spaces, it is not feasible to store a full covariance matrix. In this case, we can constrain the covariance matrix to be diagonal by truncating the off-diagonal elements of the outer product $\mathbf{s}_t \mathbf{s}_t^\top$ that appear in the update rule, eq. (5.7).

Intuitively, we expect the Laplace approximation for BLR to become increasingly accurate over time (provided that the data itself is well modeled by a logistic regression). In particular, as more examples arrive, our uncertainty in the weight vector $\mathbf{w}$ should decrease, yielding increasingly sharp posterior distributions that are very well approximated by Gaussian distributions centered at their modes.

Algorithm 1 Bayesian Logistic Regression

Parameter: $\omega > 0$

Initialize: $\boldsymbol{\mu}_0 = \mathbf{0}$, $\boldsymbol{\Sigma}_0 = \omega\mathbf{I}$

for $t = 1, 2, ...$ **do**

Receive $\mathbf{x}_t$; Predict $\hat{y}_t$; Receive y_t

$\mathbf{s}_t = \boldsymbol{\Sigma}_{t-1}\mathbf{x}_t$

Option (i): *Laplace update*

Find α_t using Newton-Raphson

$z_t = y_t\mathbf{x}_t^\top \boldsymbol{\mu}_{t-1} + \alpha_t\mathbf{x}_t^\top \mathbf{s}_t$

$\kappa_t = \sigma(z_t)\sigma(-z_t)$

$\beta_t = \kappa_t(1 + \kappa_t\mathbf{x}_t^\top \mathbf{s}_t)^{-1}$

Option (ii): *Variational update*

Find ξ_t using EM

$z_t = y_t\mathbf{x}_t^\top \boldsymbol{\mu}_{t-1}$

$\lambda_t = (\sigma(\xi_t) - \frac{1}{2})/(2\xi_t)$

$\beta_t = 2\lambda_t(1 + 2\lambda_t\mathbf{x}_t^\top \mathbf{s}_t)^{-1}$

$\alpha_t = [1 - \beta_t(\mathbf{x}_t^\top \mathbf{s}_t + 2z_t)]/2$

end

$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \alpha_t y_t \mathbf{s}_t$

$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t-1} - \beta_t \mathbf{s}_t\mathbf{s}_t^\top$ (full)

$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t-1} - \beta_t \operatorname{diag}(\mathbf{s}_t\mathbf{s}_t^\top)$ (diag)

end for

5.1.3 Variational Approximation

Alternatively, we can approximate the posterior distribution $P(\mathbf{w}|D_t)$ by a multivariate Gaussian using variational methods [JJ00]. Variational methods construct this Gaussian approximation by computing a lower bound on the integral

$$P(y_t|D_{t-1}, \mathbf{x}_t) = \int d\mathbf{w} P(\mathbf{w}|D_{t-1})\sigma(y_t\mathbf{x}_t^\top \mathbf{w}) \tag{5.8}$$

that appears in the denominator of eq. (5.3). The lower bound is obtained from a so-called auxiliary function:

$$Q(z, \xi) \ = \sigma(\xi)e^{\frac{1}{2}(z-\xi)-\lambda(\xi)(z^2-\xi^2)} \tag{5.9}$$

where $\lambda(\xi) = (\sigma(\xi) - \frac{1}{2})/(2\xi)$. Appealing to convexity properties, it can be shown that the right hand side of eq. (5.9) computes a lower bound

$$Q(z, \xi) \;\leq\; \sigma(z) \tag{5.10}$$

valid for all z and ξ, with equality holding if and only if $z = \xi$. Substituting the auxiliary function into eq. (5.8), we obtain the further bound:

$$P(y_t|D_{t-1}, \mathbf{x}_t) \;\geq\; \max_{\xi} \int d\mathbf{w}\, P(\mathbf{w}|D_{t-1})\, Q(y_t \mathbf{x}_t^\top \mathbf{w}, \xi). \tag{5.11}$$

Let ξ_t denote the value of the variational parameter that maximizes this lower bound on $P(y_t|D_{t-1}, \mathbf{x}_t)$. (An iterative EM algorithm for computing ξ_t is reviewed in Appendix B.)

Appealing to the bound in eq. (5.11), the variational approach to BLR makes the approximation:

$$P(\mathbf{w}|D_t) \;\approx\; \frac{P(\mathbf{w}|D_{t-1}) Q(y_t \mathbf{x}_t^\top \mathbf{w}, \xi_t)}{\int d\mathbf{w}\, P(\mathbf{w}|D_{t-1}) Q(y_t \mathbf{x}_t^\top \mathbf{w}, \xi_t)}. \tag{5.12}$$

Suppose that $P(\mathbf{w}|D_{t-1})$ is approximated by a Gaussian distribution with mean $\boldsymbol{\mu}_{t-1}$ and covariance matrix $\boldsymbol{\Sigma}_{t-1}$. Due to the purposely constructed form of the auxiliary function, it follows that eq. (5.12) also defines a Gaussian distribution. As shorthand, let $z_t = y_t \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1}$ and $\mathbf{s}_t = \boldsymbol{\Sigma}_{t-1} \mathbf{x}_t$. Straightforward algebra shows that the updated mean and covariance matrix of $P(\mathbf{w}|D_t)$ in the approximation are

$$\boldsymbol{\mu}_t \;=\; \boldsymbol{\mu}_{t-1} + \alpha_t y_t \mathbf{s}_t, \tag{5.13}$$

$$\boldsymbol{\Sigma}_t \;=\; \boldsymbol{\Sigma}_{t-1} - \beta_t \mathbf{s}_t \mathbf{s}_t^\top, \tag{5.14}$$

where as additional shorthand, we have introduced the notation $\beta_t = 2\lambda_t(1 + 2\lambda_t \mathbf{x}_t^\top \mathbf{s}_t)^{-1}$, $\lambda_t = \lambda(\xi_t)$ and $\alpha_t = [1 - \beta_t(\mathbf{x}_t^\top \mathbf{s}_t + 2z_t)]/2$. For prediction, we compute the tightest variational bound on $P(\hat{y}_t|D_{t-1}, \mathbf{x}_t)$ for outcomes $\hat{y}_t \in \{-1, +1\}$, then choose the label $\hat{y}_t$ that appears most likely under this approximation.

We summarize the algorithm for variational BLR in option (ii) of Algorithm 1. As in the Laplace approximation for BLR, we can constrain the covariance matrix to be diagonal by truncating non-diagonal elements in the update rule. Results from Jaakkola and Jordan [JJ00] suggest that the variational approximation

will outperform the Laplace approximation in the early stages of learning, when there is more uncertainty in the parameter estimates.

5.2 Confidence-Weighted Learning

Confidence-weighted (CW) learning is an alternative model of online learning for linear classification. As in BLR, CW also expresses its uncertainty over the decision boundary by a probability distribution over the weight vector [DCP08, CDP09]. However, the distribution in CW does not represent a posterior distribution in the traditional Bayesian sense.

CW assumes that the distribution over the weight vector is multivariate Gaussian. Let $\boldsymbol{\mu}_{t-1}$ and $\boldsymbol{\Sigma}_{t-1}$ denote the mean and covariance matrix of this distribution after observing $t-1$ labeled examples. For a new example $\mathbf{x}_t$, we predict its label by taking the sign of the expected value for the margin $\mathbf{x}_t^\top \mathbf{w}$, where $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$. The precise calculation is

$$\hat{y}_t = \text{sign}(\mathbf{x}_t^\top \boldsymbol{\mu}_{t-1}). \tag{5.15}$$

Although both CW and BLR model uncertainty in the weight vector, learning in CW is not Bayesian. In particular, the distribution over the weight vector is not updated by Bayes rule. Instead, for each labeled example $(\mathbf{x}_t, y_t)$, we solve the following optimization:

$$\begin{aligned} (\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \leftarrow \underset{\boldsymbol{\mu}, \boldsymbol{\Sigma}}{\text{argmin}} \ & \text{KL}(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \| \mathcal{N}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})) \\ \text{s.t.} \ & y_t(\boldsymbol{\mu} \cdot \mathbf{x}_t) \geq \Phi^{-1}(\eta) \sqrt{\mathbf{x}_t^\top \boldsymbol{\Sigma} \mathbf{x}_t} \end{aligned} \tag{5.16}$$

where η is a confidence parameter and Φ is the cumulative distribution function for a zero-mean Gaussian with unit variance. Intuitively, this optimization updates the Gaussian distribution over the weight vector as little as possible (i.e., minimizing the KL divergence) such that the latest example is classified correctly, by a large margin, with probability η. Note that this update aggressively updates the distribution over the weight vector to correct any margin violations that it encounters.

Algorithm 2 Confidence-Weighted Learning

Parameter: $\eta \in [0.5, 1]$
Initialize: $\boldsymbol{\mu}_0 = \mathbf{0}$, $\boldsymbol{\Sigma}_0 = \mathbf{I}$, $\phi = \Phi^{-1}(\eta)$,
$\psi = 1 + \phi^2/2$, $\xi = 1 + \phi^2$
for $t = 1, 2, \ldots$ **do**

Receive $\mathbf{x}_t$; Predict $\hat{y}_t$; Receive y_t

$\mathbf{s}_t = \boldsymbol{\Sigma}_{t-1}\mathbf{x}_t$

$v_t = \mathbf{x}_t^\top \mathbf{s}_t$

$z_t = y_t \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1}$

$\alpha_t = \max\{0, \frac{1}{v_t\xi}(-z_t\psi + \sqrt{\frac{1}{4}z_t^2\phi^4 + v_t\phi^2\xi})\}$

$u_t = \frac{1}{4}(-\alpha_t v_t \phi + \sqrt{\alpha_t^2 v_t^2 \phi^2 + 4v_t})^2$

$\beta_t = \alpha_t \phi (\sqrt{u_t} + \alpha_t \phi v_t)^{-1}$

$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \alpha_t y_t \mathbf{s}_t$

$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t-1} - \beta_t \mathbf{s}_t \mathbf{s}_t^\top$ (full)

$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t-1} - \beta_t \operatorname{diag}(\mathbf{s}_t \mathbf{s}_t^\top)$ (diag)

end for

The optimization in eq. (5.16) can be solved in closed form [CDP09], leading to the updates shown in Algorithm 2. Denoting $\mathbf{s}_t = \boldsymbol{\Sigma}_{t-1}\mathbf{x}_t$, the final updates take the simple form:

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \alpha_t y_t \mathbf{s}_t, \tag{5.17}$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t-1} - \beta_t \mathbf{s}_t \mathbf{s}_t^\top, \tag{5.18}$$

where α_t and β_t are adaptive, scalar learning rates determined by the optimization itself. Note that the update for the mean $\boldsymbol{\mu}_t$ in eq. (5.17) is proportional to $\mathbf{s}_t = \boldsymbol{\Sigma}_{t-1}\mathbf{x}_t$; in particular, the optimization reweights the example $\mathbf{x}_t$ so that highly uncertain elements of the weight vector are more aggressively updated than elements with less uncertainty in their estimates. This reweighting is perhaps the key distinguishing feature (and benefit) of CW for perceptron learning. Note also the parallel form of eq. (5.18) to the corresponding updates in eq. (5.7) and (5.14) for BLR, though the coefficients β_t are computed differently in each case.

5.3 Comparison of BLR and CW

The last two sections have reviewed the algorithms for BLR (with Laplace and variational approximations) and CW in a way that highlights their common structure. We emphasize the following similarities:

- Both model the uncertainty in the weight vector by a multivariate Gaussian distribution.
- Both update the mean of this distribution by adding a correction $\alpha_t \mathbf{s}_t$, where $\mathbf{s}_t = \boldsymbol{\Sigma}_{t-1}\mathbf{x}_t$.
- Both update the covariance matrix of this distribution by subtracting an outer product $\beta_t \mathbf{s}_t \mathbf{s}_t^\top$.

Note how in both approaches, the term $\mathbf{s}_t = \boldsymbol{\Sigma}_{t-1}\mathbf{x}_t$ plays a prominent role in the update. The magnitude of $\boldsymbol{\Sigma}_{t-1}$ measures the uncertainty or lack of confidence in different components of the weight vector $\mathbf{w}$. By rescaling each example $\mathbf{x}_t$ in this way, BLR and CW differentiate between components of $\mathbf{w}$ that have been estimated with low and high confidence. The updates for BLR and CW aggressively adapt their distributions over low-confidence components, while more cautiously adapting their distributions over high-confidence components.

Our review of BLR and CW also reveals fundamental differences. We emphasize the following contrasts:

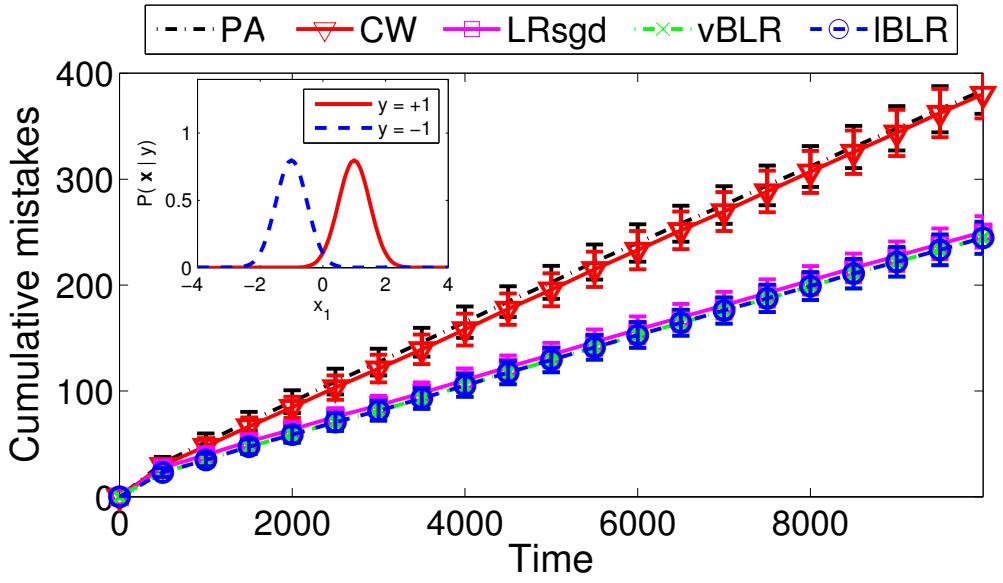
- The performance of BLR is sensitive to the initial value of the covariance matrix, as determined by the scale-parameter $\omega > 0$. The errors made by CW are invariant to the initial scale of the covariance matrix (Lemma 3 from Crammer et al. [CDP09]), but overall performance is sensitive to the pre-specified confidence parameter η.
- BLR involves an iterative, one-dimensional optimization to update its posterior distribution (in the Laplace and variational approximations), while CW has closed-form updates.
- BLR aims to maximize an approximate likelihood objective over all examples it encounters, whereas CW instantly moves to correct the most recent margin violation, thus privileging recent examples over previous ones.

While the first two differences are fairly minor computational issues, the last suggests that BLR and CW may exhibit rather different types of asymptotic behavior on large data sets. In particular, for noisy data or highly overlapping classes, we might expect that CW's aggressive tendency to correct margin violations will lead to different (and inferior) solutions over time than BLR. By contrast, for non-stationary data, we might expect this tendency of CW to reduce the cumulative number of prediction errors. The next section evaluates this hypothesis.

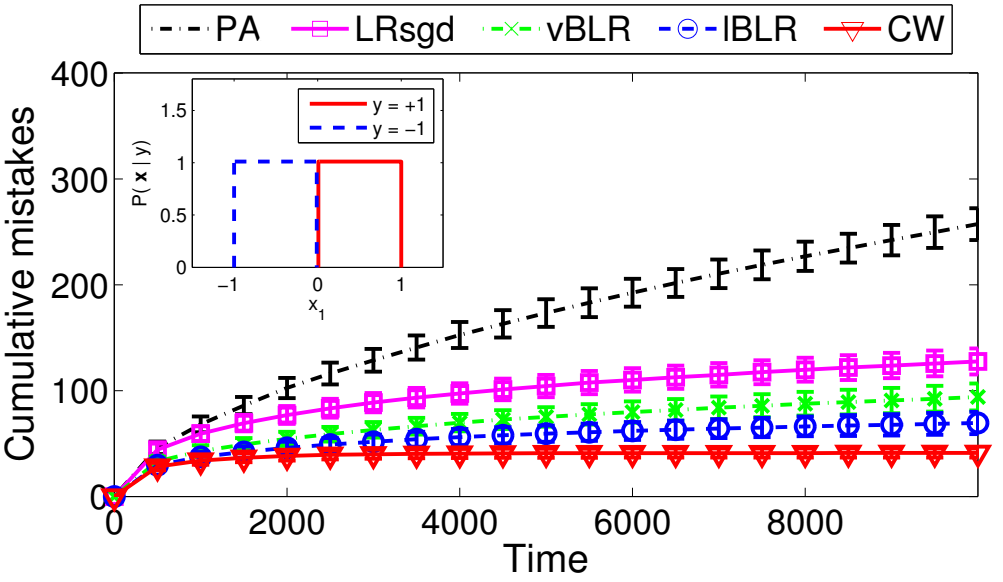
5.4 Evaluation

How do the differences between Bayesian logistic regression (BLR) and confidence-weighted (CW) learning play out in practice? In this section, we compare the accuracy of BLR and CW on several synthetic data sets as well as detecting malicious URLs. Our experiments on synthetic data sets examine how the performance of these approaches depends on certain basic or implicit assumptions. Our large-scale experiments illustrate the performance on an application of significant commercial interest.

We evaluate BLR and CW in an online scenario, permitting their models to train incrementally on a single pass over the examples in each data set. For BLR, we evaluate both the Laplace approximation (ℓBLR) and the variational approximation (vBLR). In addition to ℓBLR, vBLR, and CW, we also include results from logistic regression with stochastic gradient descent (LRsgd) and passive-aggressive (PA) learning [CDSSS06]. We view the latter as non-probabilistic counterparts to BLR and CW which do not explicitly model the uncertainty of parameter estimation. In many experiments, the contrast with these simpler approaches is also interesting. Finally, our implementations of BLR and CW use a full covariance matrix in the experiments on synthetic data sets and a diagonal covariance matrix in malicious URL detection (due to the large number of features).



(a) Gauss-1D



(b) Margin-1D

Figure 5.1: Cumulative mistakes for the Gauss-1D and Margin-1D data sets. Insets: class-conditional distributions for the first element of the feature vector in these data sets.

5.4.1 One Dimensional Separation

Our first experiments illustrate basic differences between logistic regression and large margin classification (which naturally carry over to BLR and CW). We

generate points in $\mathbb{R}^{20}$ where the distributions for positive and negative examples differ only in the first coordinate. The data is split evenly (50/50) between positive and negative examples, which are alternately presented for online learning. We consider two different types of distributions.

Gauss-1D: This experiment evaluates the sensitivity of various methods to noisy data and overlapping classes. We generate 10,000 points from two multivariate Gaussian distributions with diagonal covariance matrices. Values in the first coordinate are distributed as $\mathcal{N}(1, \frac{1}{4})$ for positive examples and $\mathcal{N}(-1, \frac{1}{4})$ for negative examples; all other features are drawn from independent zero-mean Gaussians with unit variance. The inset of Fig. 5.1(a) plots the distribution of the first coordinate for positive and negative examples. Here, we can describe the true posterior distribution $P(y|\mathbf{x})$ of this experiment by a logistic regression; thus the basic parametric assumptions of BLR are true.

To tune the parameters, we used a separate 10,000 point validation set to arrive at the following settings: a learning rate of 0.01 for LRsgd, $\omega = 0.1$ for vBLR, $\omega = 0.01$ for ℓBLR, and $\eta = 0.65$ for CW.

Fig. 5.1(a) shows the cumulative number of mistakes (y-axis) in these experiments over time, which is expressed as the number of training examples observed thus far (x-axis). Here, the results show that methods based on logistic regression (LRsgd and BLR) clearly outperform the methods based on large margin classification (PA and CW). For this data set, the latter algorithms are perturbed by occasional outliers; in particular, even though CW models uncertainty in the weight vector, it still suffers because it cannot learn the correct underlying noise model. On the other hand, these experiments do not reveal any significant differences between LRsgd versus BLR or PA versus CW.

Margin-1D: This experiment reverses the assumptions of the previous one. We generate 10,000 points from two non-overlapping uniform distributions. The first coordinate's values are drawn uniformly from $[\frac{1}{100}, 1]$ for positive examples and $[-1, -\frac{1}{100}]$ for negative examples; all other features are drawn uniformly from the interval $[-1, 1]$. The inset of Fig. 5.1(b) plots the distribution of the first coordinate for positive and negative examples. In this experiment, the data is separated by

a small but finite margin, and the true posterior distribution $P(y|\mathbf{x})$ is not well modeled by logistic regression. Thus the data violates BLR's basic parametric assumptions.

We used a separate 10,000 point validation set to get the following parameters: learning rate of 0.01 for LRsgd, $\omega = 0.1$ for vBLR, $\omega = 10$ for ℓBLR, $\eta = 0.90$ for CW.

Fig. 5.1(b) shows CW outperforming all other methods in this case. CW outperforms BLR and LRsgd because the data does not match the latter's noise model. CW outperforms PA because the latter can only update its weight vector in the direction of actual examples $\mathbf{x}_t$, most of whose elements contain noise. By contrast, CW explicitly models its uncertainty in different components of the decision boundary and learns to ignore the irrelevant features in this problem.

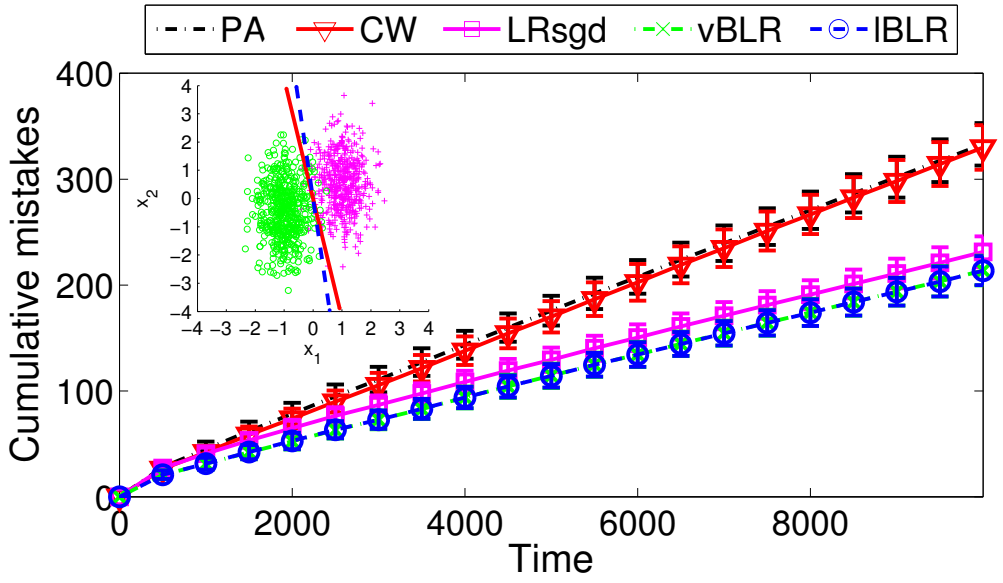
5.4.2 Two Dimensional Separation

Our next experiments illustrate the advantages of methods that model uncertainty when the data contains multiple correlated features that are relevant for classification. Again we generate points in $\mathbb{R}^{20}$, but this time the distributions for positive and negative examples differ in the first *two* coordinates. As before, the data is split 50/50 between positive and negative examples, which are alternately presented for online learning. We consider two types of distributions.

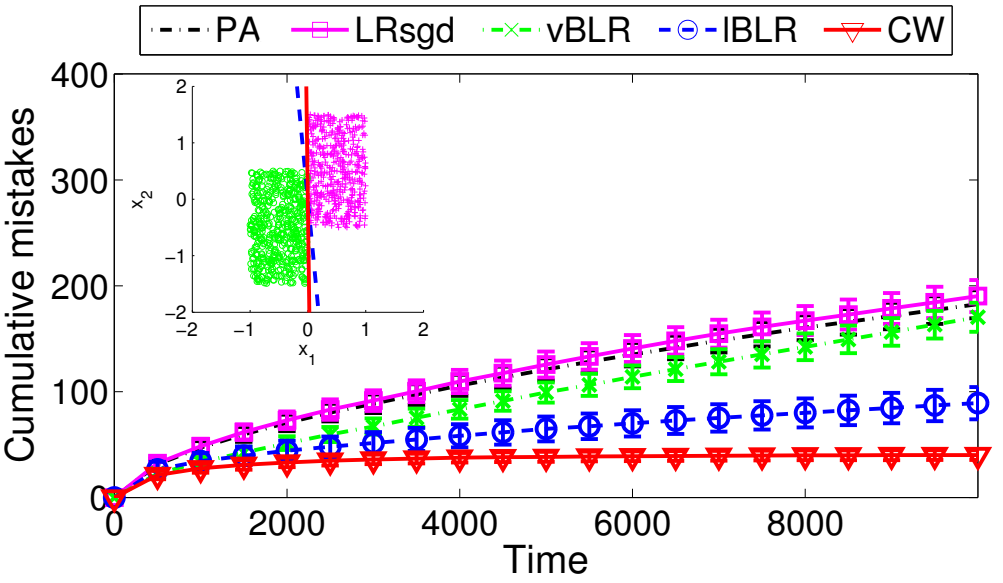
Gauss-2D: This experiment examines the performance on noisy data with multiple relevant features. The experimental parameters are the same as Gauss-1D, except that values in the second coordinate are distributed as $\mathcal{N}(\frac{1}{2}, 1)$ for positive examples and $\mathcal{N}(-\frac{1}{2}, 1)$ for negative examples. The inset of Fig. 5.2(a) plots the first two features of the data set for positive and negative examples. Again, in this experiment, the true posterior distribution $P(y|\mathbf{x})$ of the data is described by a logistic regression; thus the basic parametric assumptions of BLR are true.

We used a separate 10,000 point validation set to get the following parameters: learning rate of 0.01 for LRsgd, $\omega = 0.1$ for vBLR, $\omega = 0.1$ for ℓBLR, $\eta = 0.65$ for CW.

The results in Fig. 5.2(a) show that BLR outperforms LRsgd more clearly



(a) Gauss-2D



(b) Margin-2D

Figure 5.2: Cumulative mistakes for the Gauss-2D and Margin-2D data sets. Insets: plot of the first two elements of the feature vectors in these data sets, differently colored for positive and negative examples.

than in the Gauss-1D case. It seems that BLR's incorporation of uncertainty allows it to learn the correct decision boundary more quickly when the data contains multiple relevant features.

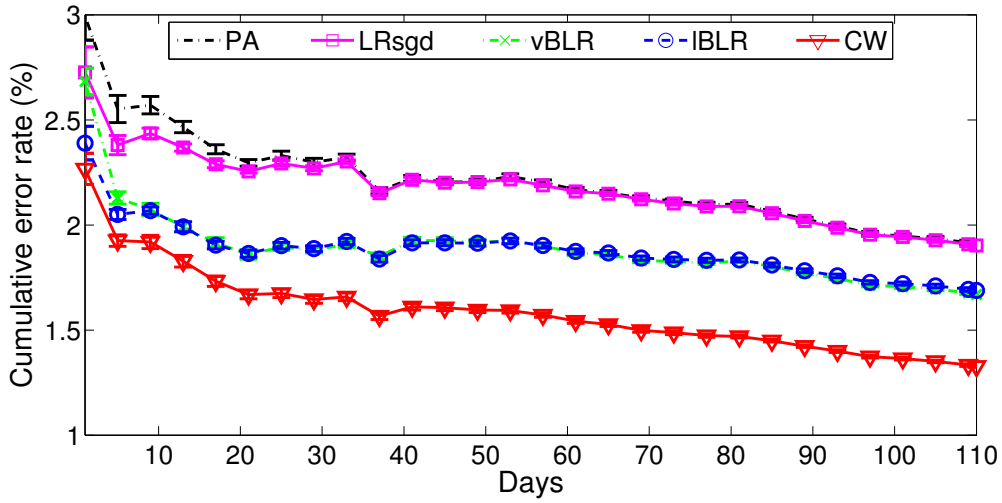
Margin-2D: This experiment examines the performance on data that contains a mix of different features which distinguish the classes to varying degrees. The experiment parameters are the same as Margin-1D, except that values in the second coordinate are drawn uniformly from the interval $[-\frac{1}{2}, \frac{3}{2}]$ for positive examples and the interval $[-\frac{3}{2}, \frac{1}{2}]$ for negative examples. The inset of Fig. 5.2(b) shows that the class-conditional distributions in this data set resemble those in Gauss-2D (over which BLR succeeded) except that they are now separated by a small margin.

We used a separate 10,000 point validation set to get the following parameters: learning rate of 1.0 for LRsgd, $\omega = 1$ for vBLR, $\omega = 100$ for ℓBLR, $\eta = 0.90$ for CW.

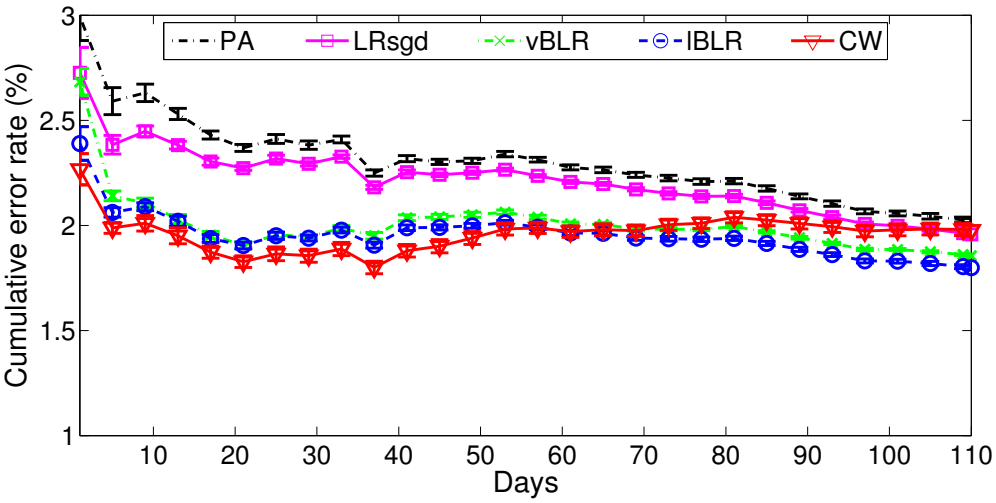
Fig. 5.2(b) shows that the performance gap between CW and other methods is slightly wider than in the one-dimensional case, especially with ℓBLR making more mistakes in Margin-2D than in Margin-1D. To visualize this result, the inset of Fig. 5.2(b) shows the decision boundaries learned by ℓBLR (dashed line) and CW (solid line). Because the data does not match ℓBLR's parametric assumptions, ℓBLR learns a decision boundary that does not separate the two classes. By contrast, CW learns to separate the two classes based on the first (separable) feature, while ignoring the second feature of the data.

Overall, these results suggest that if the data contains multiple relevant features at different scales, then BLR and CW perform better than their non-probabilistic counterparts, LRsgd and PA. We attribute this improvement to the fact that BLR and CW explicitly model the uncertainty in different components of the decision boundary and can update certain components more aggressively than others. An even more striking observation is that as long as the classes are separable (even by a small margin), then CW learns to ignore other directions in the input space along which the data is partially but not completely separated. By contrast, BLR does not focus on the margin to the exclusion of other data components.

The synthetic results suggest that CW may be especially well suited for applications that contain a mix of different features: some relevant, some irrelevant,



(a) Growing feature vector



(b) Fixed feature vector

Figure 5.3: Cumulative error rates for classifying the live, real-time URL feed using a growing vs. fixed feature vector. The minimum value of the y-axis is 1%.

some which separate the classes very well, and some which do not. For such applications, we might expect the same relative ranking of the algorithms as in Fig. 5.2(b), with CW first, followed by BLR, followed by LRsgd and PA.

5.4.3 Application: Detecting Malicious URLs

For our final experiments, we return to the dissertation's focus of detecting malicious Web sites from suspicious URLs. We use the data set from Chapter 4, and we perform experiments over 110 days of this data. To generate confidence intervals while maintaining the sequential order of the original data, we executed 10 runs of the experiment where each example had a 0.5 probability of being included in the run. Because the original URL data set consisted of 20,000 URLs per day, the effective data set size for each run was approximately 1.1 million examples.

The URLs are classified as malicious or benign based on features derived from their lexical and host-based properties. The features are computed or gathered in real time as we receive the live feed of labeled URLs. Lexical features indicate which tokens are present in the URL (represented as a bag-of-words). Host-based features are derived geographic, domain name, and IP-related properties of the URL. Binary features are associated with every geographic location, Internet service provider, and registrar/registrant that we encounter in the feed. Thus the feature set is constantly evolving; by Day 110, the URLs are represented by sparse feature vectors with over two million elements.

To tune the parameters, we used a separate validation set of 10 days of data for the following settings: a learning rate of 0.01 for LRsgd, $\omega = 1$ for vBLR, $\omega = 0.1$ for ℓBLR, and $\eta = 0.90$ for CW.

Fig. 5.3(a) shows the cumulative error rates for this experiment, and we notice three overall trends. First, the algorithms that incorporate uncertainty (BLR and CW) obtain a 12–30% improvement over PA and LRsgd, which obtain a cumulative error rate of 1.90% at best. This result is consistent with our earlier results on synthetic data sets, again suggesting that BLR and CW are more robust to the presence of noisy and/or irrelevant features.

The second trend is that CW significantly outperforms BLR throughout the entire experiment. This result suggests that the data is linearly separable, or nearly so, and that the aggressive update of CW is exploiting this property. We believe that linear separability is made possible by the new features introduced on a continual basis. To corroborate this hypothesis that new features play an

important role, we ran an additional experiment limiting the feature set to the first 150K features gathered from the initial days of the feed, as shown in Figure 5.3(b). We found that without the influx of new features over the two month period, CW eventually performed worse than BLR.

5.5 Summary

We have compared two approaches to online learning of linear classifiers: Bayesian logistic regression (BLR) versus confidence-weighted (CW) learning. Both approaches incorporate uncertainty by maintaining a probability distribution over the weight vector that defines the decision boundary. We have highlighted the parallel structure of BLR and CW algorithms, while also noting the significant differences between them. Our experiments on synthetic and real-world data sets demonstrated the complementary strengths and weaknesses of these approaches: BLR seems more appropriate for noisy data and overlapping classes, while CW is very well suited for data that is linearly separable or nearly so.

We found that CW significantly outperformed BLR for our approach to detecting malicious URLs. In our application, CW exploited the margin created by the continual introduction of new, relevant features over time. At the same time, BLR performed well compared to non-Bayesian approaches such as LRsgd and PA. Overall, our results indicate the promise of confidence-weighted methods such as BLR and CW for online learning. Although these methods share a deceptively similar structure, they differ in key details. Our experiments have helped to develop a better understanding of these differences.

Given the success of online linear classification — especially CW — for detecting malicious URLs, the question arises of how to better use correlations between features to improve classification accuracy. Up to this point, we have been modeling feature variances independently using a diagonal covariance matrix $\mathbf{\Sigma}$ for large-scale, high-dimensional applications; no doubt there are correlations among the millions of features that we can exploit as well. We explore this issue in the following chapter.

Chapter 6

Exploiting Feature Correlations

In Chapter 4, we demonstrated the successful application of online linear classification to the problem of detecting malicious URLs. In particular, the experiments showed confidence-weighted learning was the most effective algorithm; CW models feature weight uncertainty, a distinguishing aspect of the algorithm which is primarily responsible for its success. However, throughout that investigation we modeled the uncertainty of each feature weight independently, ignoring any correlations between features. In this chapter we address the following question: how does classification improve if we exploit the covariance structure of the feature space? We explore this issue in the general setting and relate the results back to our application of malicious URL classification.

6.1 Overview

Online linear classification is well-suited for learning from large, rapidly growing, high-dimensional data sets because it makes a single pass over the training data and only needs to store the current example and the current classification hypothesis. Among online linear classifier learners, those that maintain second-order information have shown special promise because second-order information can lead to faster convergence on a single pass over the training data. In confidence-weighted (CW) learning [DCP08, CDP09] and Bayesian logistic regression [JJ00, Mac92, SL90], second-order information represents uncertainty about the linear

classifier's feature weight estimates and can be modeled as a Gaussian distribution over the classifier's weight vector. The mean of the weight vector is used for classification, and the covariance matrix is used to modulate the learning rate over different features.

Unfortunately, storing and updating the full covariance matrix requires time and space quadratic in the number of features, which becomes prohibitively expensive when that number grows much beyond 10^4. Efficient diagonal approximations, which scale linearly with the number of features, are often used in practice [DCP08, CDP09, CKD09]. However, these approximations sacrifice information about cross-feature correlations that lead to faster convergence. Thus diagonal approximations trade accuracy for speed.

We investigate the nature of this tradeoff, using synthetic experiments to show when it is advantageous to use a full covariance rather than a diagonal covariance matrix. In these experiments, we consider variations in the data's dimensionality, the amount of correlation among the features, and the amount of noise in the data set. We then propose a novel method for online learning by approximating the inverse covariance matrices using a low-rank matrix. Our approach forms a practical middle-ground, improving performance over diagonal methods without incurring the high computational costs of modeling full covariance. We base our methods on the CW framework, although we believe the approach is also applicable to other covariance-tracking online algorithms, such as Bayesian logistic regression, the second-order perceptron [CBCG05] and quasi-Newton gradient descent [Bot98]. We show empirical benefits on a variety of real world data sets.

We begin with a review of the CW algorithm and compare methods for computing full and diagonal updates. Then, we introduce our inverse covariance approximation algorithm and follow with experimental results on synthetic and real-world data. We conclude with related work and discuss our findings.

6.2 Confidence-Weighted Online Learning

Online learning algorithms operate in rounds. During round t, the algorithm receives an instance $\boldsymbol{x}_t \in \mathbb{R}^d$ and applies its current rule to make a prediction $\hat{y}_t$. It then receives the true label y_t and suffers a loss $\ell(y_t, \hat{y}_t)$. Using this information, the algorithm updates its prediction rule and proceeds to the next round. The goal of the learner is to minimize its cumulative loss.

As in other chapters of this dissertation, we consider binary classification problems where $\hat{y}_t, y_t \in \{-1, +1\}$ and $\ell(y_t, \hat{y}_t) = \mathbb{I}(y_t \neq \hat{y}_t)$ is the zero-one loss function. In this case, the cumulative loss is simply the number of incorrect predictions (mistakes). To predict $\hat{y}_t$ we use a linear model parameterized by a weight vector $\boldsymbol{w} \in \mathbb{R}^d$, $\hat{y}_t = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x}_t)$.

The design of the update rule has a significant impact on performance. A simple approach is to increment the weight vector by $y_t \boldsymbol{x}_t$ whenever the loss is nonzero; this moves the score $\boldsymbol{w} \cdot \boldsymbol{x}_t$ in the right direction and yields the perceptron algorithm. A better approach in many cases is the passive-aggressive rule, which scales the perceptron update as needed to ensure that $\boldsymbol{x}_t$ is correctly classified with margin [CDSSS06].

More recently, Dredze, Crammer and Pereira proposed a new framework called confidence weighted (CW) learning that allows the update rule to consider confidence information about the model parameters [DCP08, CDP09]. Rather than maintaining a single weight vector from round to round, a CW learner maintains a Gaussian distribution over weight vectors, parameterized by a mean vector $\boldsymbol{\mu} \in \mathbb{R}^d$ and a covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$, that represents the learner's confidence about its parameter values. By accounting for the shape of this distribution, CW algorithms can make more effective updates to the weights, for example by refining them preferentially along directions that are currently low-confidence (high-variance).

At test time, one imagines drawing a weight vector from the learned distribution and then using it to make a prediction. However, for binary classification it turns out that predictions made using the mean weight vector $\boldsymbol{\mu}$ are Bayes optimal with respect to sampling $\boldsymbol{w}$ (because the Gaussian is symmetric) as well as simpler

to produce. Thus in practice the confidence information $\boldsymbol{\Sigma}$ serves primarily as a regularizer for training.

The specific update used by CW classifiers is a passive-aggressive rule modified to account for confidence information. Following round t, a weight vector drawn from the updated distribution is required to correctly classify $\boldsymbol{x}_t$ with probability at least $\eta \in (0.5, 1]$. Subject to this constraint, the algorithm makes the lowest possible KL divergence change to the hypothesis weight distribution:

$$(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) = \min_{\boldsymbol{\mu},\boldsymbol{\Sigma}} \ \mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \,\|\, \mathcal{N}\left(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\right)\right) \tag{6.1}$$

$$\text{s.t.} \ \ \Pr_{\boldsymbol{w}\sim\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})}\left[y_t\left(\boldsymbol{w}\cdot\boldsymbol{x}_t\right) \geq 0\right] \geq \eta. \tag{6.2}$$

This optimization can be solved in closed form, yielding the following update equations, called the CW-Stdev update in [CDP09]:

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \alpha_t y_t \boldsymbol{\Sigma}_t \boldsymbol{x}_t, \tag{6.3}$$

$$\boldsymbol{\Sigma}_{t+1} = \boldsymbol{\Sigma}_t - \beta_t \boldsymbol{\Sigma}_t \boldsymbol{x}_t \boldsymbol{x}_t^\top \boldsymbol{\Sigma}_t. \tag{6.4}$$

The constants α_t and β_t are nonnegative learning rates computed as given in eq. 22 of Crammer et al. [CDP09]. We note that the covariance update can alternatively be written in the inverse:

$$\boldsymbol{\Sigma}_{t+1}^{-1} = \boldsymbol{\Sigma}_t^{-1} + \frac{\alpha_t \phi}{\sqrt{u_t}} \boldsymbol{x}_t \boldsymbol{x}_t^\top, \tag{6.5}$$

as defined in eqs. 10–11 of Crammer et al. [CDP09]. This representation of the update is particularly useful for the factored inverse covariance approximation that we discuss in Section 6.3.

6.2.1 High-Dimensional Applications

Some of the most successful applications of CW learning involve high-dimensional data, especially from natural language processing [CDP09, DCP08, DC08]. Clearly, it is impractical to maintain a full d^2 covariance matrix in those cases. Instead, $\boldsymbol{\Sigma}$ is approximated with a diagonal matrix to produce an algorithm that scales linearly with the size of the feature vocabulary. An empirically

successful approximation method is to begin with a diagonal matrix and project back onto the set of diagonal matrices after each update. This projection can be done using the ℓ_2 norm, which simply drops the off-diagonal terms in eq. (6.4), or using KL-divergence, which corresponds to dropping the off-diagonal terms of $\mathbf{\Sigma}^{-1}$. Both of these approaches work well in practice, and for simplicity we proceed here using ℓ_2 projection.

6.2.2 Benefits of Full $\mathbf{\Sigma}$

In diagonal CW learning, the element $\Sigma_{p,p}$ of the covariance matrix encodes the learner's confidence in the mean weight μ_p for feature p. Given the update rule in eq. (6.4), it is easy to see that $\Sigma_{p,p}$ shrinks whenever feature p is observed in the data; this corresponds to increased confidence in μ_p and smaller subsequent updates to that value. Thus, the diagonal of $\mathbf{\Sigma}$ serves to decay the effective learning rate on a per-feature basis, hopefully leading to faster convergence.

However, the off-diagonal elements of $\mathbf{\Sigma}$, discarded by the diagonal approximation, can also provide useful guidance during training. In the following we attempt to characterize some of the ways in which full $\mathbf{\Sigma}$ can provide improved training regularization compared to diagonal $\mathbf{\Sigma}$.

Consider a pair of binary features (x_p, x_q) that co-occur frequently. We maintain a separate weight for each of these features, and given enough data a learner can estimate their values independently. However, knowing that the features are correlated, we might hope to do better by replacing them in advance with a new pair of features: $(x_p + x_q, x_p - x_q)$. While this change has no effect on the expressive power of the model (or its literal dimension), it does change its geometry: we have replaced two similar features with one more common feature and another that is usually zero.

In the context of diagonal CW learning, we are now better equipped to learn from these data. Our confidence about the weight for $x_p + x_q$ will grow more quickly than for $x_p - x_q$, because the observed values $x_p + x_q$ are far from zero more often. This enables us to quickly reduce the *effective* dimensionality of the learning problem, since we need not consider large changes to weights that are

highly confident. We no longer zig (upon seeing feature p alone) and zag (upon seeing feature q alone); instead we make consistent changes to the newly combined weights. This is analogous to the shift from gradient to conjugate gradient methods in the optimization literature [NW99, Figs. 5.1 and 5.2]. While gradient descent will zig-zag when the Hessian of the objective is non-diagonal, conjugate methods effectively diagonalize the Hessian and converge quickly and directly.

For our toy example, we have described a transformation of the features explicitly; however, similar effects can be obtained adaptively and implicitly through the use of full $\boldsymbol{\Sigma}$. While diagonal methods deal with each feature independently, full methods can tie them together, simplifying the problem of locating a good weight vector by regularizing the updates into an effectively lower-dimensional space. Especially when there are many features and relatively few examples, this can be a significant advantage. We demonstrate this effect with a simple synthetic experiment.

We begin each trial by sampling a true weight vector $\boldsymbol{w}^* \in \mathbb{R}^{10}$ from a ten-dimensional normal distribution. On round t, we flip ten coins to produce "true" binary features $\boldsymbol{z}_t \in \{-1, +1\}^{10}$ and compute the label $y_t = \text{sign}(\boldsymbol{w}^* \cdot \boldsymbol{z}_t)$. We then construct the observed features by creating k duplicates of $\boldsymbol{z}_t$ and randomly flipping 5% of the resulting binary values, producing $\boldsymbol{x}_t \in \{-1, +1\}^{10k}$. These data share many properties with the simple example discussed above.

We applied both full and diagonal CW methods to data sets of varying size for $k \in \{1, \ldots, 10\}$ and recorded the average difference in online accuracy over 100 trials. The results are shown in Fig. 6.1(a). When there are many features and few examples, full CW learning significantly outperforms the diagonal method (error bars not shown). To demonstrate the regularizing effects of full CW we plot $\text{Tr}(\boldsymbol{\Sigma}_t)/\lambda_1(\boldsymbol{\Sigma}_t)$ at each learning round t for the 50 feature case, averaged over 20 trials (Fig. 6.1(b)). Here, $\lambda_1(\boldsymbol{\Sigma}_t)$ is the largest eigenvalue of $\boldsymbol{\Sigma}_t$. We refer to this measurement as the "effective dimension" because it characterizes the eigenvalue distribution of $\boldsymbol{\Sigma}$ as being either spherical (high-dimension) or squashed (low-dimension). When the effective dimension is low, the learner has fewer degrees of freedom to update its parameters. From the figure it is clear that full CW tightens

its regularization more quickly. Note that the two methods have roughly equal $\mathrm{Tr}(\boldsymbol{\Sigma})$ at each round.

From Fig. 6.1(a) we also see that, given enough data, diagonal CW learning significantly outperforms the full version. This is because the same ability to adapt to data co-dependencies that helps full CW learning during the early rounds leads it to adapt to noise as it approaches the optimal weight vector when the data are not separable. For example, during rounds 400–500 of the 10-feature experiment (where both methods are essentially converged), the diagonal algorithm adjusts the angle of $\boldsymbol{\mu}_t$ by an average of 0.81° per round, while the full algorithm adjusts it by an average of 2.42° per round. This increased "thrashing" leads to reduced long-term performance of full CW learning.

These observations raise the question of possible intermediates between full and diagonal learning that balance fast convergence (full) and efficient, robust learning in high dimension (diagonal). We explore such a middle ground: a method that can approximate the inter-feature correlations of full CW learning but also scales well to high-dimensional data.

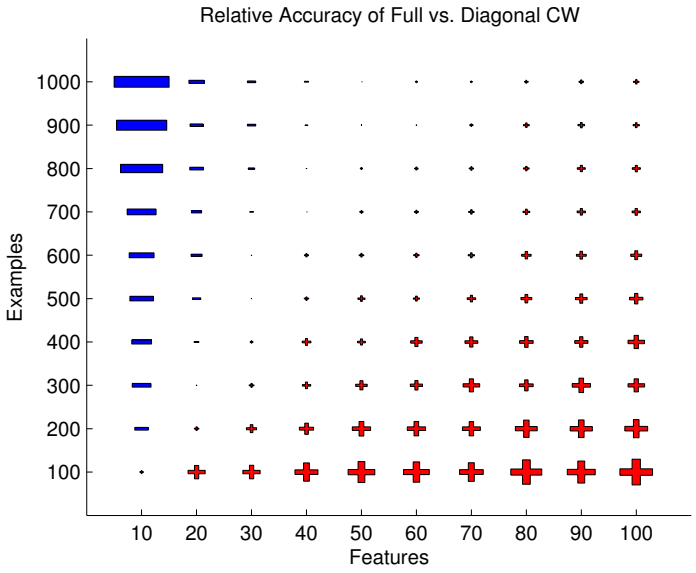
6.3 Factoring the Covariance Matrix

We observe that a matrix can be stored compactly if it is well approximated by a matrix of low rank. In our approach, we model the inverse covariance matrix for CW as the sum of a diagonal matrix plus a low rank positive semidefinite matrix, giving the factored approximation

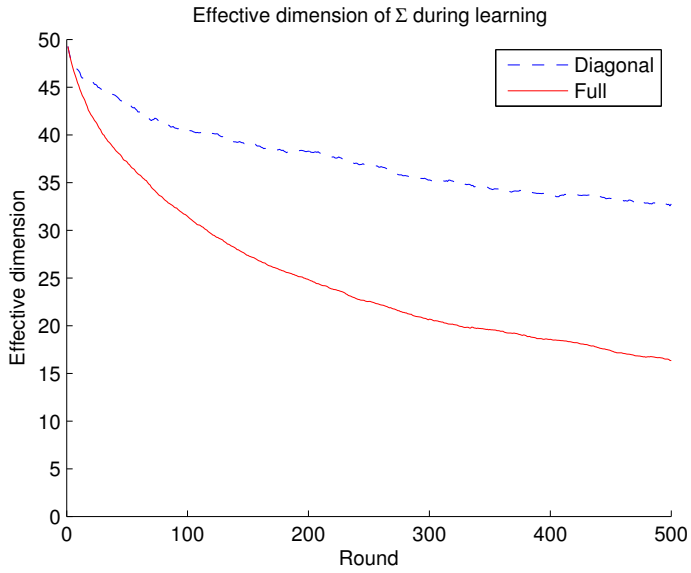
$$\boldsymbol{\Sigma}^{-1} \approx \boldsymbol{D} + \boldsymbol{R}\boldsymbol{R}^{\top}, \tag{6.6}$$

where $\boldsymbol{D}$ is a $d \times d$ diagonal matrix and $\boldsymbol{R}$ is a $d \times m$ rectangular matrix. Intuitively, this approximation is well-suited to CW because each update to the inverse covariance matrix is the addition of a vector outer product.

The approximation in eq. (6.6) is inspired by the statistical method of factor analysis [Gor83]. However, in standard factor analysis this approximation is used to model the covariance matrix, not the inverse covariance matrix; we will return to this point later. For CW learning, the approximation in eq. (6.6) has



(a) Accuracy gap of full vs. diagonal



(b) Effective dimension

Figure 6.1: (a) Average accuracy gap between full and diagonal CW, averaged over 100 trials. Pluses indicate full CW outperforming diagonal CW, minuses indicate the reverse, and the scale of each symbol depends linearly on the magnitude of the gap. The largest minus reflects a gap of -7.5%, and the largest plus reflects a gap of 4.2%. (b) Effective dimension of $\mathbf{\Sigma}$ at each round, 50 features, averaged over 20 trials.

important advantages over purely low-rank approximations (e.g., singular value decomposition) that do not include a diagonal component. First, for CW learning, we require a proper Gaussian density over the weight vector; the diagonal component in eq. (6.6) is needed to ensure that the density is normalizable. Second, the diagonal component models the per-component errors of the remaining low rank approximation, as opposed to assuming that all weights are equally uncertain. The latter assumption, though simplifying, is entirely contrary to the spirit of CW learning. Finally, as we show next, there are iterative updates for learning approximations of the form in eq. (6.6) that scale well with the dimensionality of the problem, whereas singular value decomposition may not be feasible for matrices of extremely large size. Indeed, one leading algorithm for PCA in high-dimensional spaces is an iterative estimation procedure [Row98] that can be viewed as a special case of the algorithm for maximum likelihood factor analysis. Although we focus on CW, our approach can be applied to other second-order online algorithms that need to store and maintain a positive semidefinite matrix.

6.3.1 Approximation Algorithm

Our algorithm attempts to minimize a measure of discrepancy between a target matrix $\boldsymbol{P}$ (assumed to be positive semidefinite) and its approximation $\boldsymbol{D}+\boldsymbol{R}\boldsymbol{R}^\top$. To measure discrepancy, we use the KL divergence between a pair of multivariate Gaussian distributions with the same mean (assumed without loss of generality to lie at the origin) but different covariance matrices $\boldsymbol{P}$ and $\boldsymbol{D}+\boldsymbol{R}\boldsymbol{R}^\top$:

$$\min_{\boldsymbol{D},\boldsymbol{R}} \ \mathrm{D}_{\mathrm{KL}}(\ \mathcal{N}(\mathbf{0},\boldsymbol{P})\ ||\ \mathcal{N}(\mathbf{0},\boldsymbol{D}+\boldsymbol{R}\boldsymbol{R}^\top)\ ) \tag{6.7}$$

Unlike the updates for CW learning in Section 6.2, the optimization in eq. (6.7) cannot be solved in closed form. However, we can search for a local minimum of the KL divergence by adapting the iterative updates for maximum likelihood factor analysis (see Appendix C for full derivation). As shorthand, we define the following matrices:

$$\boldsymbol{\Phi} = (\mathbf{I}+\boldsymbol{R}^\top\boldsymbol{D}^{-1}\boldsymbol{R})^{-1}, \tag{6.8}$$

$$\boldsymbol{\Upsilon} = \boldsymbol{\Phi}\boldsymbol{R}^\top\boldsymbol{D}^{-1}. \tag{6.9}$$

Note that the matrices $\boldsymbol{\Phi}$ and $\boldsymbol{\Upsilon}$ depend on the current approximation parameters, namely the rectangular matrix $\boldsymbol{R}$ and the diagonal matrix $\boldsymbol{D}$. In terms of these matrices, the updates to minimize eq. (6.7) are as follows:

$$\boldsymbol{R} \leftarrow \boldsymbol{P}\boldsymbol{\Upsilon}^\top(\boldsymbol{\Phi} + \boldsymbol{\Upsilon}\boldsymbol{P}\boldsymbol{\Upsilon}^\top)^{-1}, \tag{6.10}$$

$$\boldsymbol{D} \leftarrow \mathrm{diag}(\boldsymbol{P} - \boldsymbol{R}\boldsymbol{\Upsilon}\boldsymbol{P}). \tag{6.11}$$

To minimize eq. (6.7), we alternate between recomputing the matrices in eqs. (6.8–6.9) and updating the model parameters in eqs. (6.10–6.11). Applied in this way, the updates converge monotonically to a local minimum of the KL divergence in eq. (6.7). Note that the "target" matrix $\boldsymbol{P}$ remains fixed throughout this procedure.

6.3.2 Integration with CW Learning

The algorithm in Section 6.3.1 integrates naturally with CW learning to provide a compact approximation for full covariance matrices. We refer to this approach as *CW-fact*. To avoid performing the relatively expensive minimization procedure described in eqs. (6.8–6.11) following every update, we augment our approximation with a buffer:

$$\boldsymbol{\Sigma}^{-1} = \boldsymbol{D} + \boldsymbol{R}\boldsymbol{R}^\top + \mathbf{B}\mathbf{B}^\top, \tag{6.12}$$

where $\mathbf{B}$ is a $d \times m$ matrix holding up to m of the most recent exact updates. We update the buffer $\mathbf{B}$ after each example, but fit the factored model using eqs. (6.8–6.11) only when the buffer becomes full.

We initialize $\boldsymbol{D}$ to the identity matrix, and $\boldsymbol{R}$ and $\boldsymbol{B}$ to zero. The first m updates to $\boldsymbol{\Sigma}$ dictated by the CW algorithm are stored directly in $\boldsymbol{R}$. Using the inverse rule in eq. (6.5), the tth column of $\boldsymbol{R}$ is given by $(\frac{\alpha_t \phi}{\sqrt{u_t}})^{\frac{1}{2}} \boldsymbol{x}_t$. The next m updates fill $\boldsymbol{B}$ in the same way. When the buffer is full, we run the algorithm in Section 6.3.1 to compress the contents of both $\boldsymbol{R}$ and $\boldsymbol{B}$ into $\boldsymbol{D}$ and $\boldsymbol{R}$ (where we set the target matrix to $\boldsymbol{P} = \boldsymbol{D} + \boldsymbol{R}\boldsymbol{R}^\top + \mathbf{B}\mathbf{B}^\top$). This leaves the buffer empty, and the cycle of filling the buffer and compressing repeats for the remainder of the examples.

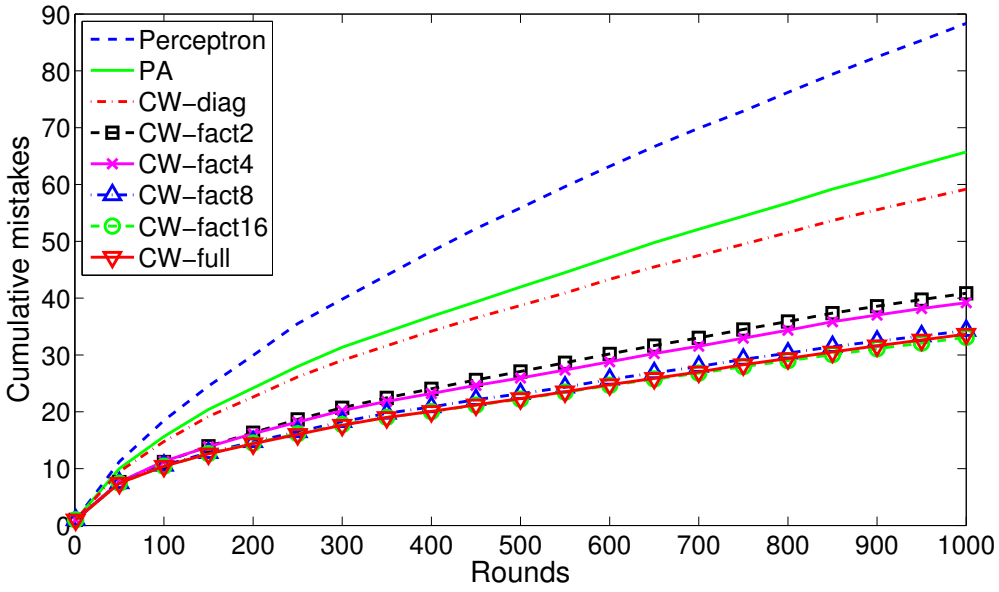


Figure 6.2: Cumulative error over the synthetic data for CW-fact(m), CW-full and CW-diag.

6.3.3 Comparison with Full and Diagonal Covariance Representations

We begin our empirical results by demonstrating that CW-fact occupies a middle ground between CW-full and CW-diag on the synthetic data described in Section 6.2. This time we generate 1000 examples with 1000 features ($k = 100$).

Figure 6.2 shows the cumulative mistake counts averaged over 100 runs. We include perceptron and passive-aggressive results for reference. As we increase m, the accuracy of CW-fact approaches CW-full. (CW-fact's performance did not improve beyond $m = 16$, which makes sense given the ten-dimensional underlying distribution.) Table 6.1 shows the average runtime and memory overhead for the different variations of CW in this experiment. The memory usage for CW-fact is an order of magnitude improvement over CW-full, and the runtime is 5× faster. Thus, CW-fact provides an adjustable compromise between the high-accuracy of CW-full and the low-overhead of CW-diag.

Table 6.1: Runtime and memory benchmarks for the synthetic experiment in Figure 6.2.

	Time (s)	**Time w/o buffer (s)**	**Mem (KB)**
CW-diag	0.09	—	7.81
CW-fact2	1.61	2.61	87.21
CW-fact4	1.35	4.11	181.27
CW-fact8	1.21	7.16	370.15
CW-fact16	1.50	16.45	750.90
CW-full	7.00	—	7812.50

6.3.4 Benefits of Buffering and $\boldsymbol{\Sigma}^{-1}$

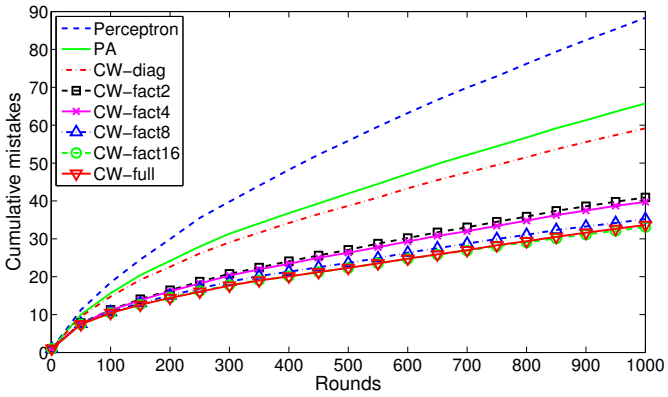
Before we move on, it is worth briefly addressing the effect of the buffer matrix on the performance of CW-fact. Figure 6.3(a) shows the results of the same synthetic experiment when we eliminate the buffer. The difference in accuracy from the experiment in Figure 6.2 is negligible. However, Table 6.1 shows that the computational cost is quite high.

We also tested the performance of buffering alone; that is, simply throwing out the oldest update whenever a new one arrives. The results are in Figure 6.3(b). They show that while buffering alone offers some benefit, it is less effective than compressing information into $\boldsymbol{R}$ to provide a long-term summary of updates to $\boldsymbol{\Sigma}^{-1}$, as done for CW-fact.

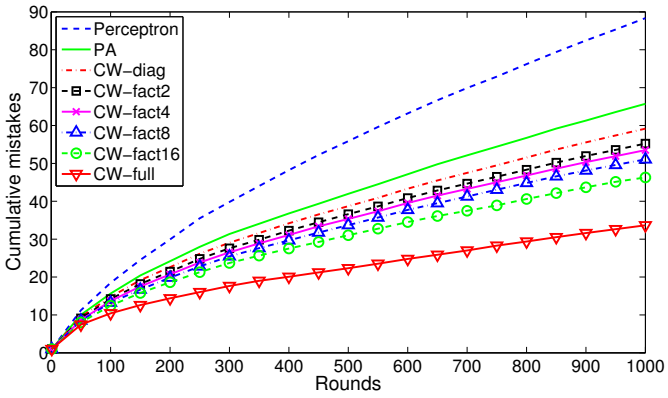
Finally, the algorithm in Section 6.3.1 appears to be a good fit for approximating $\boldsymbol{\Sigma}^{-1}$, since the update (6.5) is additive, matching the form of the approximation. However, we could also consider approximating $\boldsymbol{\Sigma}$ in the same way. Figure 6.3(c) shows that performance is dramatically reduced in this case, possibly because the approximation is a poor fit for the subtractive $\boldsymbol{\Sigma}$ update (6.4).

6.4 Large-Scale Learning

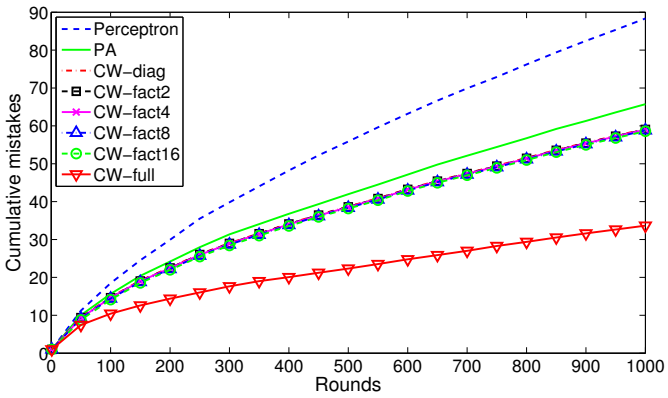
We evaluate CW learning with our factored approximation (CW-fact) on several high-dimensional real-world data sets where the number of features exceeds the number of examples and many features are correlated. We use perceptron,



(a) $\mathbf{\Sigma}^{-1}$, no buffer



(b) $\mathbf{\Sigma}^{-1}$, buffering only



(c) $\mathbf{\Sigma}$, no buffer

Figure 6.3: Evaluating alternative design decisions for CW-fact with respect to buffering and whether to approximate $\mathbf{\Sigma}$ vs. $\mathbf{\Sigma}^{-1}$. Results for perceptron, PA, CW-diag and CW-full are repeated to provide a reference point.

passive-aggressive (PA) learning [CDSSS06], and diagonal CW as baselines. For perceptron and PA, we make predictions at each round using the average of all previous weight vectors, which tends to outperform the single most recent weight vector.

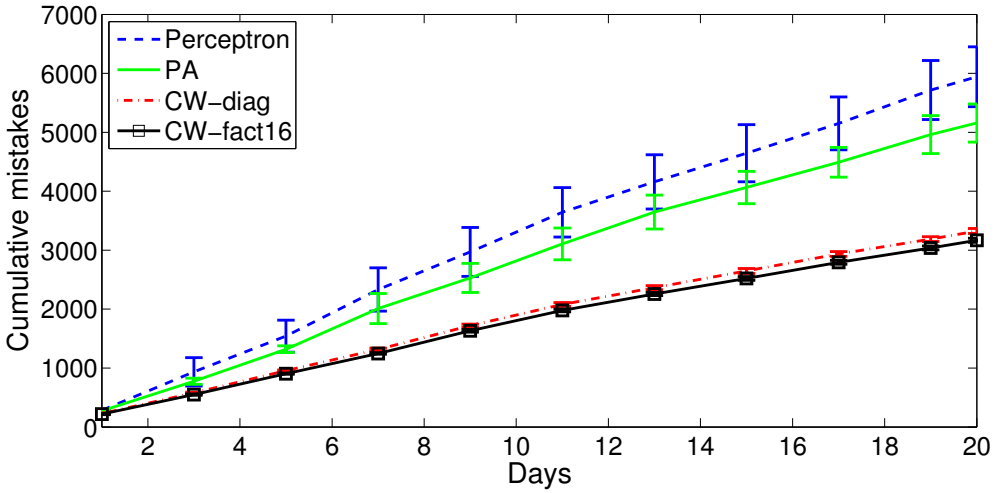
6.4.1 Detecting Malicious URLs

We evaluate CW-fact on a 20-day subset of the data set from Chapter 4, which has about 1 million binary features and 64 real-valued features scaled to the interval $[0, 1]$. The goal is to determine whether each Web site is malicious; the ratio of positive to negative examples is $1 : 2$. There are 20,000 examples collected per day. We subsample the examples, preserving the temporal ordering, to produce error bars: in each run an example has a 50% chance of being included in the evaluation set (resulting in 10,000 examples per day). Results are computed on 10 samples of 200,000 examples each.

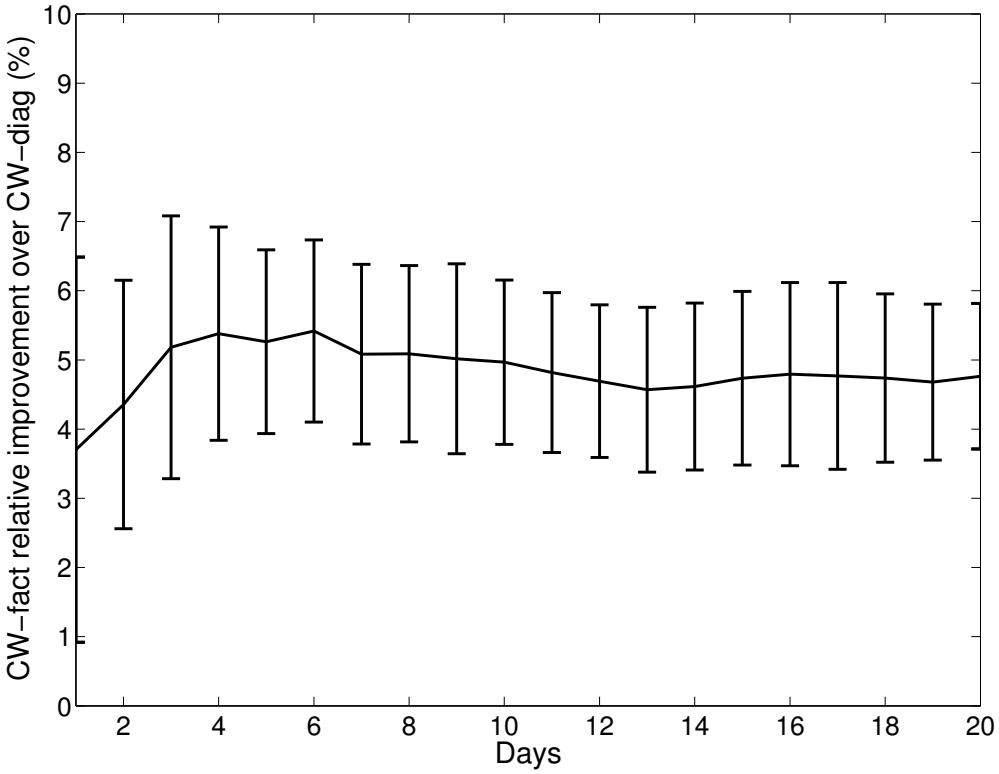
Fig. 6.4 shows the absolute and relative mistake counts (CW-diag vs. CW-fact) over time for CW-diag and CW-fact as well as perceptron and a passive-aggressive (PA) learner [CDSSS06]. CW-fact provides a consistent, 5% relative improvement over CW-diag, which is itself superior to PA and perceptron. This result corroborates our findings in Section 6.3, which showed improvements when there are more features than examples and many features are correlated.

6.4.2 Web Spam

Next, we consider the Web spam data from the PASCAL Large Scale Learning Challenge [SFYTS08]. The task is to classify Web pages from search engine results as being spam or non-spam. We divide the data set into 35 epochs containing 10,000 examples each. Over 10 trials, we include an example in the evaluation data with probability 0.5 (on average, 5,000 examples per epoch). This results in 680,000 features and an average of 175,000 examples per run. The default representation contains trigram counts, which were normalized so that each example had unit length.



(a) Mistake count



(b) Relative improvement

Figure 6.4: URL Data: (a) Mistakes made by each algorithm over 20 days. (b) Relative improvement in error rate of CW-fact over CW-diag. Error bars are at one standard deviation.

Results (Fig. 6.5) are similar to the URL experiment: CW-fact improves over CW-diag significantly (about 18% relative). Again, this data set has more features than examples and various sets of features are correlated, for example trigrams with shared bigrams or unigrams.

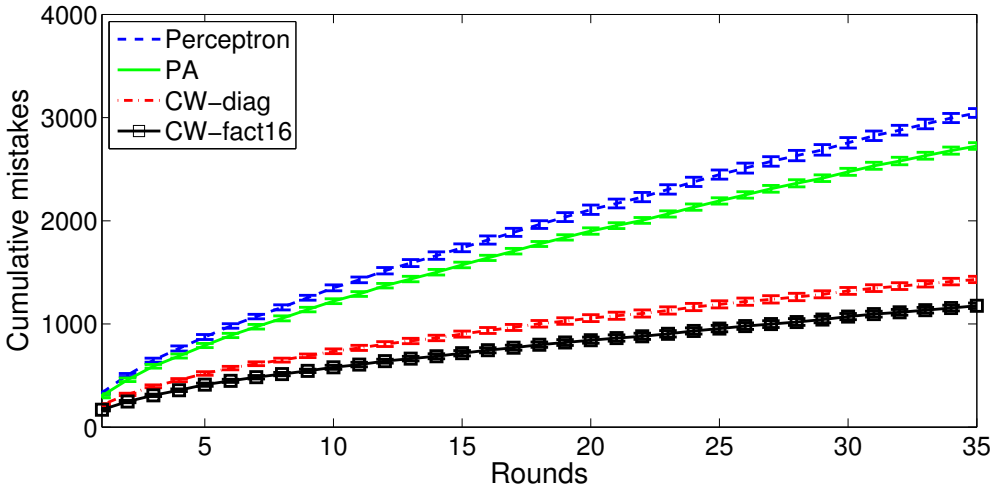
6.4.3 Stock Market Data

Given computational resource constraints, the choice of learning method effectively determines the size of the feature set that can be used in practice. CW-diag allows very large sets, while CW-full imposes relatively harsh limits. We show here, using a stock market prediction task, that CW-fact offers an advantageous middle ground: performance losses from approximating the covariance matrix can be compensated by improvements from the use of a more informative feature set, giving the highest overall performance with limited resources.

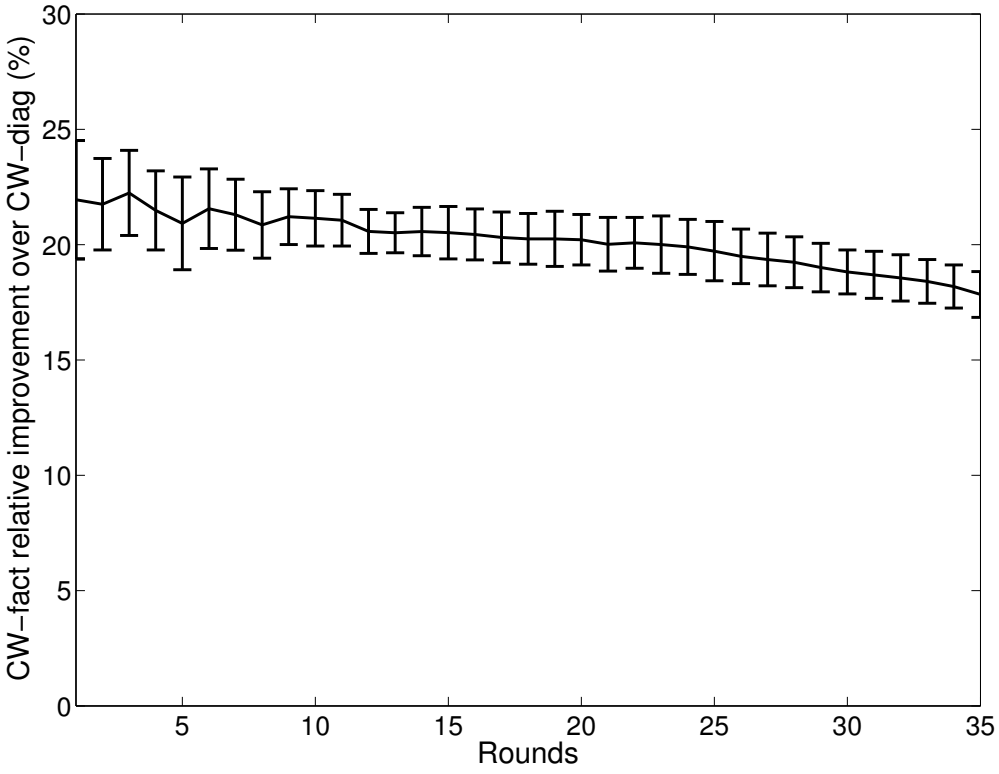
The task is to predict whether the price of a target stock went up or down each day, based on the open, high, low, and close prices of a set of predictor stocks for that day and the preceding 49 days (200 features per predictor). The feature set is thus highly correlated. Our data spans a 15-year period from 1994 to 2009, for a total of 4006 instances (2120 up days and 1886 down days). We use DELL as a target, although other tested targets showed similar results. The number of predictor stocks is variable, offering a tradeoff between speed and accuracy.

Table 6.2 shows the number of mistakes made by each method for a range of predictor set sizes, subject to a memory constraint of 2GB and a time constraint of two hours. Each result is the mean over 10 random draws of the predictor stocks from our collection of 378, excluding DELL.

CW-diag handles all feature set sizes, but is unable to extract a useful signal; note that perceptron and PA returned similarly poor results on these data (not shown). CW-full extracts the most performance from each feature set, but incurs large computational costs, requiring 54 minutes and 200MB of memory for the 25-predictor test. Completing the 100-predictor test would have required over 3GB of memory and more than 10 hours. CW-fact, on the other hand, requires only 38MB of memory and 63 minutes to run with 250 predictors, and achieves



(a) Mistake count



(b) Relative improvement

Figure 6.5: Web Spam Data: (a) Mistakes made by each algorithm (10k examples per epoch.) (b) Relative improvement in error rate of CW-fact over CW-diag. Error bars are at one standard deviation.

Table 6.2: Number of mistakes on stock market data and relative improvements over CW-diag. Values are omitted for any test that required more than 2GB of memory (†) or two hours of runtime (∗). All differences of at least 44 mistakes are statistically significant at $p = 0.01$ based on a paired t-test.

Predictors	**CW-diag**	**CW-fact8 (rel. improvement)**	**CW-full (rel. improvement)**
1	2141.0	2083.1 (2.7%)	2028.3 (5.3%)
10	2142.0	2036.8 (4.9%)	1891.4 (11.7%)
25	2142.0	1949.3 (9.0%)	1806.3 (15.7%)
100	2142.0	1779.0 (17.0%)	†∗
250	2142.0	1749.9 (18.3%)	†∗
378	2142.0	∗	†∗

the overall best result.

6.4.4 Document Classification

Lastly, we evaluate CW-fact on the document classification data sets used earlier to evaluate CW learning [DCP08]. The `Reuters` and `20 Newsgroups` data sets are binary classification tasks that require us to distinguish two closely related topics within each set (using binary bag-of-words features). The `Sentiment` set is a sentiment classification task that requires us to distinguish positive from negative reviews for different Amazon product categories (using unigram and bigram counts). For each data set, we performed 10 runs of the experiment where we randomized the order of the examples. The results in Table 6.3 show that CW-fact consistently improves over CW-diag over all sets.

6.5 Summary and Related Work

We examined the effect of covariance matrix representation on confidence-weighted learning, but we believe our results have the potential to generalize to other second-order online learning algorithms. Depending on properties of the data, full covariance or an approximate covariance matrix obtained by factored representations may improve on the more efficient diagonal covariance version of CW learning.

A desire for compact representations of second-order information arises in contexts outside of our own work. For instance, in the limited memory BFGS method (L-BFGS) for quasi-Newton algorithms, the Hessian is computed based on the last m updates [LN89]. This approach is similar in spirit to our approach of buffering the last m updates as described in Section 6.3.2. Other techniques such as Kronecker factorization and incomplete Cholesky factorization have been explored in the context of approximating kernel matrices for support-vector machine training [WCCH06].

In synthetic experiments and large-scale applications, we showed that full and factored representations performed better than diagonal when there were many correlated features and the effective dimensionality of the data was small. Conversely, we also showed that full methods performed worse when the data was noisy or had fewer correlations between features. Thus, in the context of detecting malicious URLs, approximating the full covariance matrix shows promise in further improving the classification accuracy of our approach.

Chapter 6, in part, is a reprint of the material as it appears in Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS) 2010. Ma, Justin; Kulesza, Alex; Dredze, Mark; Crammer, Koby; Saul, Lawrence K.; Pereira, Fernando. The dissertation author was a primary investigator and author of this paper.

Table 6.3: Document Classification: Average error rates (%) and standard deviations for perceptron, PA, CW-diag and CW-fact8 on `Reuters`, `20 Newsgroups` and `Sentiment` binary classification tasks.

Task	Examples	Features	Perceptron	PA	CW-diag	CW-fact8
Reuters						
business	2000	12167	24.7 ± 0.8	21.4 ± 0.7	18.7 ± 0.6	**18.1** ± 0.5
insurance	2000	9094	19.7 ± 0.4	16.8 ± 0.7	13.7 ± 0.6	**12.7** ± 0.5
retail	2000	8768	28.8 ± 0.8	26.3 ± 0.7	20.9 ± 0.7	**18.4** ± 0.5
20 Newsgroups						
comp	1943	29409	28.2 ± 1.8	21.3 ± 0.5	13.6 ± 0.5	**12.3** ± 0.6
sci	1971	38699	20.1 ± 0.8	15.1 ± 0.5	8.1 ± 0.4	**7.1** ± 0.4
talk	1850	44397	19.0 ± 1.7	11.1 ± 0.3	4.7 ± 0.3	**3.7** ± 0.4
Sentiment						
apparel	1940	63088	25.8 ± 0.5	19.5 ± 0.4	17.5 ± 0.5	**16.8** ± 0.4
books	18391	1042928	22.3 ± 0.2	17.8 ± 0.2	14.7 ± 0.1	**14.3** ± 0.1
dvd	13152	850348	23.5 ± 0.2	19.0 ± 0.2	15.8 ± 0.2	**15.4** ± 0.2
electronics	5774	249863	23.6 ± 0.2	18.6 ± 0.3	16.1 ± 0.3	**15.5** ± 0.3
kitchen	5212	193467	23.2 ± 0.4	17.5 ± 0.3	15.3 ± 0.2	**14.8** ± 0.2
music	12927	666927	24.3 ± 0.2	19.8 ± 0.3	16.3 ± 0.1	**15.7** ± 0.2
video	4349	346659	27.9 ± 0.6	22.7 ± 0.3	20.1 ± 0.4	**19.4** ± 0.4

Chapter 7

Conclusion

Over the past two decades, the World Wide Web has become a bedrock of contemporary society. From commerce to government to research, Web sites have become a vital and convenient way for people to conduct business, gather new information, discover new worlds and perform myriad other functions — all without the need to leave their seat. The convenience of the Web has had immeasurable impact in improving our lives; it has also left us immeasurably vulnerable.

Without the typical cues associated with physical world interactions, it becomes easy for people to overlook the dangers of visiting a malicious Web site. In the physical world, we observe clues, read news and talk with friends to assess the reputation of places and neighborhoods we want to visit. By contrast, the Web lacks clear "rules of thumb" for whether a site is safe to visit — and criminals have been quick to exploit this situation. Web sites for spam-advertised commerce, financial fraud (such as phishing) and propagating malware have become platforms for criminal enterprises. The mere click of a malicious site's Uniform Resource Locator (URL) is enough to lead users to harm.

The research community has responded to these threats by developing blacklisting services that compile a list of "known bad" sites and client-side systems that analyze Web sites as they are visited. These services are helpful, but also have their disadvantages. Blacklists are only useful for detecting known threats and lag behind the endless stream of newly-introduced malicious sites. Client-side systems have high run-time overhead and may inadvertently expose their users to

the threats we seek to avoid. To address the limitations of existing approaches, we developed an approach that was both scalable and adaptive.

7.1 Impact

The central thesis of this dissertation is that we can mitigate the disadvantages of blacklists and client-side systems by constructing an adaptable, lightweight URL classification system. Before, this approach to malicious URL detection generally involved using a small, hand-picked set of features and batch machine learning algorithms. However, we demonstrated that our approach of using a larger feature set had clear advantages over manual feature set construction (Chapter 3). This approach involved automatically enumerating every lexical token, network location and other host-based features we encountered. The results of our evaluations have convinced some researchers, who were initially hesitant about the seemingly indiscriminate feature gathering, to adopt our approach in their own systems.

Moreover, we explained the benefits of online learning over standard batch learning for malicious URL detection (Chapter 4). We constructed a system for real-time feature collection and classification and showed that training on as large a set of contemporaneous data as possible resulted in greater detection accuracy — especially compared to training on a limited data set and training on old data. Online learning overcame the limitations of batch learning because of its ability to process large-scale data sets with fewer computational resources and to incorporate fresh training data incrementally. In a future where there will be more (not less) data at our disposal, our results demonstrated that scalable and adaptive detection is feasible, representing a significant advance over previous work. As a result, industrial organizations have adopted our techniques for their own malicious URL detection systems.

In the course of tackling the URL classification problem, we explored important questions that arose out of our use of online learning. We addressed the differences between confidence-weighted (CW) learning and Bayesian logistic re-

gression (BLR) in Chapter 5. CW was our algorithm of choice in Chapter 4, whereas BLR was an older technique that warranted consideration. Both incorporated per-feature-weight uncertainty to achieve better accuracies than standard online algorithms, but we demonstrated that CW was more suitable for our application because its aggressive large-margin criteria was better suited for adapting to the myriad new features that arise during the course of daily training.

Finally, we examined the benefits and tradeoffs of exploiting feature correlations in high-dimensional online learning in Chapter 6. We demonstrated that for malicious URL detection and many related applications, approximating the full covariance matrix for second-order online algorithms led to improved classification accuracy.

7.2 Future Work

To succeed, our approach to malicious URL detection relies heavily on informative features and large sources of up-to-date training data. There are a number of ways to expand the work of this dissertation, and two fundamental directions include (a) providing a larger arsenal of distinguishing features and (b) finding alternative large-scale sources of labeled data. Here, we describe salient examples of gathering new features (content-based classification) and more labeled data (user feedback).

7.2.1 Content-Based Classification

An open question concerns how much additional accuracy we can achieve if we incorporate content-based features of the Web site. Indeed, the incorporation of content opens a realm of possible techniques to explore. Up to this point, we have used linear classification because it scales easily to large-scale problems. However, content contains high-level structure and semantics that are potentially useful but difficult to exploit using linear classification over low-level features.

One promising approach is to incorporate topic model features for linear classification. Topic model features can describe the probability that the content

of a page belongs to different categories — e.g., the number of topic features would be equal to the number of possible categories, each having a real value between 0 and 1. In the related field of phishing email classification, adding topic features yielded significant improvements [BCP+08].

However, with topic models we would face scalability and adaptability challenges similar to those we faced with linear classification. Is it sufficient to train a topic model once, or are there significant advantages (both in terms of speed and accuracy) to learning the topics in an online fashion? Instead of using a fixed number of topics, can we benefit from using a nonparametric model where we allow the number of topics to grow over time?

7.2.2 User Feedback

At present, our approach to detecting malicious URLs relies on a streaming source of expert-labeled examples. Instead, can we ask for labels from the users themselves to construct an effective classifier? We envision a deployment of our URL classifier — e.g., as a browser toolbar — where users can provide explicit or implicit feedback on whether a site they visit is malicious. However, trusting the users presents interesting challenges:

- Should we incorporate user input into the classifier right away, or should user input be delayed?
- Will we need to employ a reputation system for users to distinguish users who provide accurate labels from those who provide inaccurate labels (e.g., via collaborative filtering)?
- Can we find an appropriate balance between soliciting explicit feedback (annoying for users, but accurate) and implicit feedback (invisible to users, but potentially inaccurate)?
- Do we give more weight to custom per-user classifiers, or a global classifier?
- Can we respect the privacy of the users who submit labeled URLs without sacrificing accuracy of the classifier? (URLs themselves could possibly encode

personal information.)

Addressing these challenges would empower community-based URL security efforts by providing an abundant source of labeled URLs to the research community and the public.

7.3 Final Thoughts

Looking forward, we expect the synthesis of systems and machine learning to play an important role in future research challenges. Real systems can provide a rich source of large-scale data. Because large-scale data can be difficult to analyze manually, machine learning provides a methodical way to understand and build models of that data. However, real systems are often governed by resource and time constraints; an uninformed application of high-overhead, resource-intensive algorithms risks violating those constraints.

As a result, domain expertise and an awareness of systems-level constraints are crucial in developing a machine learning approach to solving large-scale problems. This dissertation, with its focus on detecting malicious URLs, is a successful application of this insight.

Appendix A

Laplace Approximation

We briefly describe the method for computing α_t in the Laplace approximation for BLR; see Algorithm 1 from Chapter 5.1. Let $\mathbf{w}^*$ denote the mode of the posterior distribution, which maximizes eq. (5.4). The exponent of eq. (5.4) is the following:

$$F(\mathbf{w}) = \log \sigma(y_t x_t^\top \mathbf{w}) - \frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_{t-1})^\top \boldsymbol{\Sigma}_{t-1}^{-1}(\mathbf{w} - \boldsymbol{\mu}_{t-1}). \tag{A.1}$$

To locate the maximum of $F(\mathbf{w})$, we compute its gradient and set it equal to zero. The gradient is

$$F'(\mathbf{w}^*) = y_t \sigma(-y_t \mathbf{x}_t^\top \mathbf{w}^*)\mathbf{x}_t - \boldsymbol{\Sigma}_{t-1}^{-1}(\mathbf{w}^* - \boldsymbol{\mu}_{t-1}). \tag{A.2}$$

Setting the right hand side to zero and rearranging the terms gives the expression

$$\mathbf{w}^* = \boldsymbol{\mu}_{t-1} + \sigma(-y_t \mathbf{x}_t^\top \mathbf{w}^*) y_t \boldsymbol{\Sigma}_{t-1} \mathbf{x}_t. \tag{A.3}$$

From the above, it follows that the solution for the maximum takes the form $\mathbf{w}^* = \boldsymbol{\mu}_{t-1} + \alpha_t y_t \mathbf{s}_t$, where $\mathbf{s}_t = \boldsymbol{\Sigma}_{t-1}\mathbf{x}_t$ and α_t is a scalar prefactor to be determined. To determine α_t, we substitute this form back into the function $F(\mathbf{w})$ and rewrite it as:

$$F(\alpha_t) = \log \sigma(y_t \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1} + \alpha_t \mathbf{x}_t^\top \mathbf{s}_t) - \frac{1}{2}\alpha_t^2 \mathbf{x}^\top \mathbf{s}_t. \tag{A.4}$$

We use a simple, one-dimensional Newton-Raphson procedure to maximize $F(\alpha_t)$ with respect to α_t.

Appendix B

Variational Approximation

Here, we describe the iterative procedure for computing the variational parameter ξ_t that yields the tightest bound in eq. (5.11). We initialize $\xi_t = y_t \mathbf{x}_t^\top \boldsymbol{\mu}_{t-1}$ and assume that $P(\mathbf{w}|D_{t-1}) \approx \mathcal{N}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$.

The iterative procedure for optimizing ξ_t can be viewed as an EM algorithm. The E-step of the EM algorithm computes an intermediate Gaussian approximation $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ to the true posterior distribution $P(\mathbf{w}|D_t)$. For fixed ξ_t, the mean $\boldsymbol{\mu}_t$ and covariance matrix $\boldsymbol{\Sigma}_t$ of this intermediate Gaussian approximation are computed from the updates in eqs. (5.13–5.14).

The M-step of the EM algorithm re-estimates ξ_t by the value that maximizes the auxiliary function:

$$\int d\mathbf{w} P(\mathbf{w}|D_t) \log[P(\mathbf{w}|D_{t-1}) Q(y_t \mathbf{x}_t^\top \mathbf{w}, \xi_t)]. \tag{B.1}$$

For fixed $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$, the update that maximizes eq. (B.1) is given by:

$$\xi_t \leftarrow \sqrt{\mathbf{x}_t^\top \boldsymbol{\Sigma}_t \mathbf{x}_t + (\mathbf{x}_t^\top \boldsymbol{\mu}_t)^2}. \tag{B.2}$$

The EM algorithm alternates between the E-step to re-estimate $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ and the M-step to re-estimate ξ_t. This procedure provably converges to a local maximum of the auxiliary function in eq. (B.1).

In practice, the EM algorithm can be further simplified by eliminating the variable $\boldsymbol{\Sigma}_t$ from intermediate steps of this procedure. Using the notation in Sec-

tion 5.1.3, the simplified iterative procedure can be written as:

$$\beta_t \leftarrow 2\lambda(\xi_t)(1 + 2\lambda(\xi_t)\mathbf{x}_t^\top \mathbf{s}_t)^{-1} \tag{B.3}$$

$$\alpha_t \leftarrow [1 - \beta_t(\mathbf{x}_t^\top \mathbf{s}_t + 2z_t)]/2 \tag{B.4}$$

$$\boldsymbol{\mu}_t \leftarrow \boldsymbol{\mu}_{t-1} + \alpha y_t \mathbf{s}_t \tag{B.5}$$

$$\xi_t \leftarrow \sqrt{\mathbf{x}_t^\top \mathbf{s}_t(1 - \beta_t \mathbf{x}_t^\top \mathbf{s}_t) + (\mathbf{x}_t^\top \boldsymbol{\mu}_t)^2} \tag{B.6}$$

The above scheme converges very quickly. A few iterations appear to suffice.

Appendix C

Factored Approximation

Given a Gaussian distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{P})$, we would like to use factor analysis to approximate its covariance by $\boldsymbol{P} \approx \boldsymbol{D} + \boldsymbol{R}\boldsymbol{R}^\top$ where $\boldsymbol{D}$ is a diagonal $d \times d$ noise matrix, and $\boldsymbol{R}$ the $d \times m$ factor loading matrix ($m \ll d$). Intuitively, this approach can produce good approximations if there is an inherently low-dimensional (but noisy) process that is generating high-dimensional outputs $\boldsymbol{\theta}$. We explain our EM approach more formally as follows.

Given $\boldsymbol{P}$, we want to find $\boldsymbol{D}$ and $\boldsymbol{R}$ that minimize the KL divergence between two Guassians:

$$\mathrm{D}_{\mathrm{KL}}(\ \mathcal{N}(\mathbf{0}, \boldsymbol{P})\ ||\ \mathcal{N}(\mathbf{0}, \boldsymbol{D}+\boldsymbol{R}\boldsymbol{R}^\top)\ ) = \int d\boldsymbol{\theta} P(\boldsymbol{\theta}|\boldsymbol{P}) \log \frac{P(\boldsymbol{\theta}|\boldsymbol{P})}{P(\boldsymbol{\theta}|\boldsymbol{D}, \boldsymbol{R})}, \tag{C.1}$$

where $P(\boldsymbol{\theta}|\boldsymbol{P}) = \mathcal{N}(\mathbf{0}, \boldsymbol{P})|_{\boldsymbol{\theta}}$ and $P(\boldsymbol{\theta}|\boldsymbol{D}, \boldsymbol{R}) = \mathcal{N}(\mathbf{0}, \boldsymbol{D} + \boldsymbol{R}\boldsymbol{R}^\top)|_{\boldsymbol{\theta}}$. This is equivalent to maximizing the following expression with respect to $\boldsymbol{D}$ and $\boldsymbol{R}$:

$$\int d\boldsymbol{\theta} P(\boldsymbol{\theta}|\boldsymbol{P}) \log P(\boldsymbol{\theta}|\boldsymbol{D}, \boldsymbol{R}). \tag{C.2}$$

We can model the process of generating observations $\boldsymbol{\theta}$ as a latent variable model with parameters $\boldsymbol{D}$, $\boldsymbol{R}$ and hidden variable $\boldsymbol{z}$. That is, given a vector $\boldsymbol{z}$ drawn from a standard normal distribution in m-dimensional space, we project it into d-dimensions using $\boldsymbol{R}$ and add component-wise noise that has covariance $\boldsymbol{D}$. Formally, $P(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})|_{\boldsymbol{z}}$ and $P(\boldsymbol{\theta}|\boldsymbol{z}, \boldsymbol{D}, \boldsymbol{R}) = \mathcal{N}(\boldsymbol{R}\boldsymbol{z}, \boldsymbol{D})|_{\boldsymbol{\theta}}$.

Because of this latent variable formulation, we can use the EM algorithm to maximize eq. (C.2) where the observed variable $\boldsymbol{\theta}$ has the distribution $P(\boldsymbol{\theta}|\boldsymbol{P})$.

E-step: For notational convenience, let us define

$$\begin{aligned} \boldsymbol{\Phi} &= (\boldsymbol{I} + \boldsymbol{R}^\top \boldsymbol{D}^{-1} \boldsymbol{R})^{-1}, \\ \boldsymbol{\Upsilon} &= \boldsymbol{\Phi} \boldsymbol{R}^\top \boldsymbol{D}^{-1}. \end{aligned} \tag{C.3}$$

Then for the E-step, we compute the statistics of $P(\boldsymbol{z}|\boldsymbol{\theta}, \boldsymbol{D}, \boldsymbol{R})$, the posterior distribution, as follows:

$$\begin{aligned} E[\boldsymbol{z}|\boldsymbol{\theta}, \boldsymbol{D}, \boldsymbol{R}] &= \boldsymbol{\Upsilon}\boldsymbol{\theta}, \\ E[\boldsymbol{z}\boldsymbol{z}^\top|\boldsymbol{\theta}, \boldsymbol{D}, \boldsymbol{R}] &= \boldsymbol{\Phi} + \boldsymbol{\Upsilon}\boldsymbol{\theta}\boldsymbol{\theta}^\top\boldsymbol{\Upsilon}^\top. \end{aligned} \tag{C.4}$$

M-step: Next, we want to choose new parameters $\widetilde{\boldsymbol{D}}$, $\widetilde{\boldsymbol{R}}$ that will maximize the following auxiliary function:

$$\int d\boldsymbol{\theta} P(\boldsymbol{\theta}|\boldsymbol{P}) \int d\boldsymbol{z} P(\boldsymbol{z}|\boldsymbol{\theta}, \boldsymbol{D}, \boldsymbol{R}) \log P(\boldsymbol{\theta}, \boldsymbol{z}|\widetilde{\boldsymbol{D}}, \widetilde{\boldsymbol{R}}). \tag{C.5}$$

Note that eq. (C.5) is a lower bound for eq. (C.2). When we expand the log term of eq. (C.5), we get the following:

$$\begin{aligned} & \log P(\boldsymbol{\theta}, \boldsymbol{z}|\widetilde{\boldsymbol{D}}, \widetilde{\boldsymbol{R}}) \\ = \; & \log P(\boldsymbol{\theta}|\boldsymbol{z}, \widetilde{\boldsymbol{D}}, \widetilde{\boldsymbol{R}}) P(\boldsymbol{z}) \\ = \; & \log \left[\frac{\exp\left\{-\frac{1}{2}(\boldsymbol{\theta} - \widetilde{\boldsymbol{R}}\boldsymbol{z})^\top \widetilde{\boldsymbol{D}}^{-1} (\boldsymbol{\theta} - \widetilde{\boldsymbol{R}}\boldsymbol{z})\right\}}{\sqrt{(2\pi)^d |\widetilde{\boldsymbol{D}}|}} \frac{\exp\left\{-\frac{1}{2}\boldsymbol{z}^\top \boldsymbol{z}\right\}}{\sqrt{(2\pi)^m}} \right] \\ = \; & \frac{1}{2}\left[\log |\widetilde{\boldsymbol{D}}^{-1}| - \boldsymbol{\theta}^\top \widetilde{\boldsymbol{D}}^{-1} \boldsymbol{\theta} + 2\boldsymbol{\theta}^\top \widetilde{\boldsymbol{D}}^{-1} \widetilde{\boldsymbol{R}} \boldsymbol{z} - \boldsymbol{z}^\top (I + \widetilde{\boldsymbol{R}}^\top \widetilde{\boldsymbol{D}}^{-1} \widetilde{\boldsymbol{R}}) \boldsymbol{z} \right] + c, \end{aligned} \tag{C.6}$$

where c is a constant. When we integrate eq. (C.6) with respect to $\boldsymbol{z}$, we get the following expression (we omit terms that are constant with respect to $\widetilde{\boldsymbol{D}}$ and $\widetilde{\boldsymbol{R}}$):

$$\begin{aligned} & \int d\boldsymbol{z} P(\boldsymbol{z}|\boldsymbol{\theta}, \boldsymbol{D}, \boldsymbol{R}) \tfrac{1}{2}\left[\log |\widetilde{\boldsymbol{D}}^{-1}| - \boldsymbol{\theta}^\top \widetilde{\boldsymbol{D}}^{-1} \boldsymbol{\theta} + 2\boldsymbol{\theta}^\top \widetilde{\boldsymbol{D}}^{-1} \widetilde{\boldsymbol{R}} \boldsymbol{z} - \boldsymbol{z}^\top \widetilde{\boldsymbol{R}}^\top \widetilde{\boldsymbol{D}}^{-1} \widetilde{\boldsymbol{R}} \boldsymbol{z} \right] \\ = \; & \frac{1}{2}\left[\log |\widetilde{\boldsymbol{D}}^{-1}| - \boldsymbol{\theta}^\top \widetilde{\boldsymbol{D}}^{-1} \boldsymbol{\theta} + 2\boldsymbol{\theta}^\top \widetilde{\boldsymbol{D}}^{-1} \widetilde{\boldsymbol{R}} \boldsymbol{\Upsilon} \boldsymbol{\theta} - \mathrm{tr}[\widetilde{\boldsymbol{R}}^\top \widetilde{\boldsymbol{D}}^{-1} \widetilde{\boldsymbol{R}} (\boldsymbol{\Phi} + \boldsymbol{\Upsilon}\boldsymbol{\theta}\boldsymbol{\theta}^\top \boldsymbol{\Upsilon}^\top)] \right]. \end{aligned} \tag{C.7}$$

Then, when we integrate eq. (C.7) with respect to $\boldsymbol{\theta}$ we get the following:

$$\begin{aligned} & \int d\boldsymbol{\theta} P(\boldsymbol{\theta}|\boldsymbol{P}) \\ & \tfrac{1}{2}\left[\log |\widetilde{\boldsymbol{D}}^{-1}| - \boldsymbol{\theta}^\top \widetilde{\boldsymbol{D}}^{-1} \boldsymbol{\theta} + 2\boldsymbol{\theta}^\top \widetilde{\boldsymbol{D}}^{-1} \widetilde{\boldsymbol{R}} \boldsymbol{\Upsilon} \boldsymbol{\theta} - \mathrm{tr}[\widetilde{\boldsymbol{R}}^\top \widetilde{\boldsymbol{D}}^{-1} \widetilde{\boldsymbol{R}} (\boldsymbol{\Phi} + \boldsymbol{\Upsilon}\boldsymbol{\theta}\boldsymbol{\theta}^\top \boldsymbol{\Upsilon}^\top)] \right] \\ = \; & \frac{1}{2}\left[\log |\widetilde{\boldsymbol{D}}^{-1}| - \mathrm{tr}(\widetilde{\boldsymbol{D}}^{-1} \boldsymbol{P}) + 2\mathrm{tr}(\widetilde{\boldsymbol{D}}^{-1} \widetilde{\boldsymbol{R}} \boldsymbol{\Upsilon} \boldsymbol{P}) - \mathrm{tr}[\widetilde{\boldsymbol{R}}^\top \widetilde{\boldsymbol{D}}^{-1} \widetilde{\boldsymbol{R}} (\boldsymbol{\Phi} + \boldsymbol{\Upsilon}\boldsymbol{P}\boldsymbol{\Upsilon}^\top)] \right] \\ = \; & F(\widetilde{\boldsymbol{R}}, \widetilde{\boldsymbol{D}}^{-1}), \end{aligned} \tag{C.8}$$

where we define $F(\widetilde{\boldsymbol{R}}, \widetilde{\boldsymbol{D}}^{-1})$ for notational convenience. To maximize F, we differentiate with respect to $\widetilde{\boldsymbol{R}}$ and $\widetilde{\boldsymbol{D}}^{-1}$ and set the derivatives to zero. Differentiating and solving for $\widetilde{\boldsymbol{R}}$ yields the following update for $\boldsymbol{R}$:

$$\begin{aligned} & \tfrac{\partial F}{\partial \widetilde{\boldsymbol{R}}} = \widetilde{\boldsymbol{D}}^{-1}\boldsymbol{P}\boldsymbol{\Upsilon}^\top - \widetilde{\boldsymbol{D}}^{-1}\widetilde{\boldsymbol{R}}(\boldsymbol{\Phi} + \boldsymbol{\Upsilon}\boldsymbol{P}\boldsymbol{\Upsilon}^\top) = 0 \\ \implies & \widetilde{\boldsymbol{D}}^{-1}\widetilde{\boldsymbol{R}}(\boldsymbol{\Phi} + \boldsymbol{\Upsilon}\boldsymbol{P}\boldsymbol{\Upsilon}^\top) = \widetilde{\boldsymbol{D}}^{-1}\boldsymbol{P}\boldsymbol{\Upsilon}^\top \\ \implies & \boxed{\boldsymbol{R} \leftarrow \boldsymbol{P}\boldsymbol{\Upsilon}^\top(\boldsymbol{\Phi} + \boldsymbol{\Upsilon}\boldsymbol{P}\boldsymbol{\Upsilon}^\top)^{-1}.} \end{aligned} \tag{C.9}$$

Using the updated $\boldsymbol{R}$ in place of $\widetilde{\boldsymbol{R}}$, we can differentiate by $\widetilde{\boldsymbol{D}}^{-1}$ to yield an update for $\boldsymbol{D}$ as follows:

$$\begin{aligned} & \tfrac{\partial F}{\partial \widetilde{\boldsymbol{D}}^{-1}} = \tfrac{1}{2}\left[\widetilde{\boldsymbol{D}} - \boldsymbol{P} + 2\boldsymbol{R}\boldsymbol{\Upsilon}\boldsymbol{P} - \boldsymbol{R}(\boldsymbol{\Phi} + \boldsymbol{\Upsilon}\boldsymbol{P}\boldsymbol{\Upsilon}^\top)\boldsymbol{R}^\top\right] = 0 \\ \implies & \widetilde{\boldsymbol{D}} = \boldsymbol{P} - 2\boldsymbol{R}\boldsymbol{\Upsilon}\boldsymbol{P} + \boldsymbol{R}(\boldsymbol{\Phi} + \boldsymbol{\Upsilon}\boldsymbol{P}\boldsymbol{\Upsilon}^\top)\boldsymbol{R}^\top \\ \implies & \widetilde{\boldsymbol{D}} = \boldsymbol{P} - 2\boldsymbol{R}\boldsymbol{\Upsilon}\boldsymbol{P} + \boldsymbol{R}(\boldsymbol{\Phi} + \boldsymbol{\Upsilon}\boldsymbol{P}\boldsymbol{\Upsilon}^\top)(\boldsymbol{\Phi} + \boldsymbol{\Upsilon}\boldsymbol{P}\boldsymbol{\Upsilon}^\top)^{-1}\boldsymbol{\Upsilon}\boldsymbol{P} \\ \implies & \widetilde{\boldsymbol{D}} = \boldsymbol{P} - 2\boldsymbol{R}\boldsymbol{\Upsilon}\boldsymbol{P} + \boldsymbol{R}\boldsymbol{\Upsilon}\boldsymbol{P} \\ \implies & \boxed{\boldsymbol{D} \leftarrow \operatorname{diag}(\boldsymbol{P} - \boldsymbol{R}\boldsymbol{\Upsilon}\boldsymbol{P}).} \end{aligned} \tag{C.10}$$

The $\boldsymbol{D}$ update can be efficient because the diagonal for $\boldsymbol{P}$ only needs to be computed once before starting the EM algorithm. Furthermore, we can reuse the expression $\boldsymbol{P}\boldsymbol{\Upsilon}^\top$, which was computed during the $\boldsymbol{R}$ update.

Bibliography

[AFSV07] David S. Anderson, Chris Fleizach, Stefan Savage, and Geoffrey M. Voelker. Spamscatter: Characterizing Internet Scam Hosting Infrastructure. In *Proc. of the USENIX Security Symposium*, Boston, MA, August 2007.

[Aga] Against Intuition. WOT Web of Trust. http://www.mywot.com.

[ANNWN07] Saeed Abu-Nimeh, Dario Nappa, Xinlei Wang, and Suku Nair. A Comparison of Machine Learning Techniques for Phishing Detection. In *Proc. of the Anti-Phishing Working Group eCrime Researchers Summit*, Pittsburgh, PA, October 2007.

[BCP+08] André Bergholz, Jeong-Ho Chang, Gerhard Paaß, Frank Reichartz, and Siehyun Strobel. Improved Phishing Detection using Model-Based Features. In *Proc. of the Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, August 2008.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, 2006.

[BL04] Léon Bottou and Yann LeCun. Large Scale Online Learning. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[BLFM98] Tim Berners-Lee, Roy Fielding, and Larry Masinter. Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396, August 1998.

[Bot98] Léon Bottou. Online Algorithms and Stochastic Approximations. In *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

[BS94] J. M. Bernardo and A. F. M. Smith. *Bayesian Theory*. John Wiley & Sons, New York, NY, 1994.

[CBCG05] Nicoloò Cesa-Bianchi, Alex Conconi, and Claudio Gentile. A Second-Order Perceptron Algorithm. *SIAM Journal on Computing*, 34(3):640–668, 2005.

[CDP09] Koby Crammer, Mark Dredze, and Fernando Pereira. Exact Convex Confidence-Weighted Learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[CDSSS06] Koby Crammer, Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

[CKD09] Koby Crammer, Alex Kulesza, and Mark Dredze. Adaptive regularization of weight vectors. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[CL] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A Library for Support Vector Machines. http://www.csie.ntu.edu.tw/ cjlin/libsvm/.

[Dai04] Leslie Daigle. WHOIS Protocol Specification. RFC 3912, September 2004.

[DC08] Mark Dredze and Koby Crammer. Online Methods for Multi-Domain Learning and Adaptation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2008.

[DCP08] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-Weighted Linear Classification. In *International Conference on Marchine Learning (ICML)*, Helsinki, Finland, July 2008.

[DH98] Stephen Deering and Robert Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, December 1998.

[Dig10] Digital Element. NetAcuity. http://www.digital-element.com/ip_intelligence/ip_intelligence.html, 2010.

[Dro97] Ralph Droms. Dynamic Host Configuration Protocol. RFC 2131, March 1997.

[DSSS08] Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. The Forgetron: A Kernel-Based Perceptron on a Budget. *SIAM Journal on Computing*, 37(5):1342–1372, 2008.

[FCH+08] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A Library for Large Linear Classification. http://www.csie.ntu.edu.tw/ cjlin/liblinear/, 2008.

[FL06] V. Fuller and T. Li. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632, August 2006.

[FST07] Ian Fette, Norman Sadeh, and Anthony Tomasic. Learning to Detect Phishing Emails. In *Proc. of the International World Wide Web Conference (WWW)*, Banff, Alberta, Canada, May 2007.

[GCL09] D. J. Guan, Chia-Mei Chen, and Jia-Bin Lin. Anomaly Based Malicious URL Detection in Instant Messaging. In *Proceedings of the Joint Workshop on Information Security (JWIS)*, Kaohsiung, Taiwan, August 2009.

[Goo] Google. Google Toolbar. http://tools.google.com/firefox/toolbar/.

[Gor83] Richard L. Gorsuch. *Factor Analysis*. Lawrence Erlbaum Associates, New York, NY, 2nd edition, 1983.

[GPCR07] Sujata Garera, Niels Provos, Monica Chew, and Aviel D. Rubin. A Framework for Detection and Measurement of Phishing Attacks. In *Proc. of the ACM Workshop on Rapid Malcode (WORM)*, Alexandria, VA, November 2007.

[GRS96] Walter R. Gilks, Sylvia Richardson, and David J. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, London, 1996.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Publishing Company, New York, NY, 2001.

[Iro08] IronPort. IronPort Web Reputation: Protect and Defend Against URL-Based Threat. *IronPort White Paper*, 2008.

[JJ00] Tommi S. Jaakkola and Michael I. Jordan. Bayesian Parameter Estimation via Variational Methods. *Statistics and Computing*, 10:25–37, 2000.

[JS10] Mark Jeftovic and David Saez. PHPWhois. http://sourceforge.net/projects/phpwhois/, 2010.

[Kei08] Gregg Keizer. Spam Plummets After Calif. Hosting Service Shuttered. http://www.computerworld.com/s/article/9119963/Spam_plummets_after_Calif._hosting_service_shuttered, November 2008.

[KFJ06] Pranam Kolari, Tim Finin, and Anupam Joshi. SVMs for the Blogosphere: Blog Identification and Splog Detection. In *Proc. of the AAAI Spring Symposium on Computational Approaches to Analysing Weblogs*, Stanford, CA, March 2006.

[KT05] Min-Yen Kan and Hoang Oanh Nguyen Thi. Fast Webpage Classification Using URL Features. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, Bremen, Germany, October 2005.

[LN89] Dong C. Liu and Jorge Nocedal. On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical Programming*, 45(3):503–528, 1989.

[Mac92] David J. C. MacKay. The Evidence Framework Applied to Classification Networks. *Neural Computation*, 4(5):720–736, 1992.

[MBD+07] Alexander Moshchuk, Tanya Bragin, Damien Deville, Steven D. Gribble, and Henry M. Levy. SpyProxy: Execution-based Detection of Malicious Web Content. In *Proc. of the USENIX Security Symposium*, Boston, MA, August 2007.

[MBGL06] Alexander Moshchuk, Tanya Bragin, Steven D. Gribble, and Henry M. Levy. A Crawler-Based Study of Spyware on the Web. In *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, San Diego, CA, February 2006.

[McA] McAfee. SiteAdvisor. http://www.siteadvisor.com.

[McM08] Robert McMillan. Zeus Botnet Dealt a Blow as ISP Troyak Knocked Out. http://www.itworld.com/government/100020/zeus-botnet-dealt-blow-isp-troyak-knocked-out, November 2008.

[MG08] D. Kevin McGrath and Minaxi Gupta. Behind Phishing: An Examination of Phisher Modi Operandi. In *Proc. of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, San Francisco, CA, April 2008.

[Moc87a] Paul Mockapretis. Domain Names – Concepts and Facilities. RFC 1034, November 1987.

[Moc87b] Paul Mockapretis. Domain Names – Implementation and Specification. RFC 1035, November 1987.

[Net] Netscape. DMOZ Open Directory Project. http://www.dmoz.org.

[NW99] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, 1999.

[NWC+07] Yuan Niu, Yi-Min Wang, Hao Chen, Ming Ma, and Francis Hsu. A Quantitative Study of Forum Spamming Using Context-based Analysis. In *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, San Diego, CA, March 2007.

[OKC08] Francesco Orabona, Joseph Keshet, and Barbar Caputo. The Projectron: A Bounded Kernel-Based Perceptron. In *International Conference on Machine Learning (ICML)*, Helsinki, Finland, July 2008.

[Ope] OpenDNS. PhishTank. http://www.phishtank.com.

[PMRM08] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All Your iFRAMEs Point to Us. In *Proc. of the USENIX Security Symposium*, San Jose, CA, July 2008.

[Ros58] Frank Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6):386–408, 1958.

[Row98] Sam Roweis. EM Algorithms for PCA and SPCA. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 626–632. MIT Press, Cambridge, MA, 1998.

[SFYTS08] Soeren Sonnenburg, Vojtech Franc, Elad Yom-Tov, and Michele Sebag. PASCAL Large Scale Learning Challenge. http://largescale.first.fraunhofer.de/workshop/, July 2008.

[SL90] David J. Spiegelhalter and Steffen L. Lauritzen. Sequential Updating of Conditional Probabilities on Directed Graphical Structures. *Networks*, 20(5):579–605, 1990.

[SPS07] Fei Sha, Albert Park, and Lawrence K. Saul. Multiplicative Updates for L_1-Regularized Linear and Logistic Regression. In *Proc. of the Symposium on Intelligent Data Analysis (IDA)*, Ljubljana, Slovenia, September 2007.

[SS02] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, 2002.

[Uni10] University of Oregon Advanced Network Technology Center. Route Views Project. http://www.routeviews.org, 2010.

[USC81] USC Information Sciences Institute. Internet Protocol: DARPA Internte Program Protocol Specification. RFC 791, September 1981.

[Voe99] Geoffrey M. Voelker. Universal Resource Locator. In John G. Webster, editor, *Wiley Encyclopedia of Electrical and Electronics Engineering*, volume 23, pages 35–39. John Wiley & Sons, New York, NY, 1999.

[WBJ+06] Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King. Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities. In *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, San Diego, CA, February 2006.

[WCCH06] Gang Wu, Edward Chang, Yen-Kuang Chen, and Christopher Hughes. Incremental Approximate Matrix Factorization for Speeding up Support Vector Machines. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, Philadelphia, PA, August 2006.

[Web] WebSense. ThreatSeeker Network. http://www.websense.com/content/Threatseeker.aspx.

[WRN10] Colin Whittaker, Brian Ryner, and Marria Nazif. Large-Scale Automatic Classification of Phishing Pages. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2010.

[ZHC07] Yue Zhang, Jason Hong, and Lorrie Cranor. CANTINA: A Content-Based Approach to Detecting Phishing Web Sites. In *Proc. of the International World Wide Web Conference (WWW)*, Banff, Alberta, Canada, May 2007.

[ZPU08] Jian Zhang, Phillip Porras, and Johannes Ullrich. Highly Predictive Blacklisting. In *Proc. of the USENIX Security Symposium*, San Jose, CA, July 2008.