Florian Klien, BSc

# Leveraging Content-independent Features for Spam Detection in URL Shorteners

**Master's Thesis**

Graz University of Technology

Knowledge Technologies Institute
Head: Univ.-Prof. Dr. Stefanie Lindstaedt

Supervisor: Univ.-Doz. Dipl.-Ing. Dr.techn. Markus Strohmaier

Graz, April 2013

## Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

## Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

# Abstract

URL Shorteners have become very popular in the last few years. They take a long URL and return a short equivalent, that can be used instead. Visitors of this short-link first visit the URL Shortener and are then redirected to the original long URL. The biggest services redirect millions of users every day to various websites. But it is their basic functionality that is a source for exploits. Spammers take advantage of short-links to hide malicious websites. It is the responsibility of the URL Shortener to identify these links and to disable them. Often a URL Shortener is the first to come in contact with a spam link, rendering traditional methods, such as blacklists, unusable.

This work presents a method to identify spam without the aid of blacklists or the shortened websites' content. Analyzing a dataset that was generated by operating a URL Shortener for 21 months shows that there is useful information to help fighting spam in URL Shorteners. By only taking data that is directly available to the shortener, such as usage data, one can successfully identify spam with an accuracy of up to 97%. There is no need to crawl the shortened website or query a blacklist. This work should give a good insight into how one could implement a spam detection mechanism for URL Shorteners without the need to rely on external sources.

The results of this work are relevant for scientists, who have an interest in spam detection and practitioners, who want to prevent content-based exploits of existing spam detection systems.

# Kurzfassung

URL-Verkürzer haben in den letzten Jahren sehr an Popularität gewonnen. Benutzer können dort einen langen Link in einen kurzen Link umwandeln. Dieser kann dann an Stelle des langen Links eingesetzt werden. Die größten Dienste leiten täglich Millionen Menschen auf diverse Webseiten weiter. Es ist aber die grundlegende Funktionsweise der URL-Verkürzer, die eine Sicherheitslücke darstellt. Spammer nutzen URL-Verkürzer aus, um bösartige Links zu verstecken. Es liegt in der Verantwortung der URL-Verkürzer, diese Links zu finden und zu deaktivieren. Häufig sind URL-Verkürzer allerdings die ersten die solche Links sehen, was klassische Methoden, wie schwarze Listen, unbrauchbar macht.

Diese Arbeit stellt eine Methode vor, mit der Spam ohne Zuhilfenahme von externen Inhalten oder schwarzen Listen detektiert werden kann. Das Betreiben eines eigenen URL-Verkürzers über 21 Monate und die Analyse des resultierenden Datensatzes ergibt, dass man wertvolle Information aus lokalen Daten extrahieren kann. Mit solche Daten, wie etwa dem Benutzerverhalten, kann eine Spamdetektion mit einer Genauigkeit von bis zu 97% durchgeführt werden. Diese Arbeit soll einen Einblick geben, wie man diese neue Methode in URL-Verkürzern einsetzen kann, ohne sich auf externe Daten verlassen zu müssen.

Die Resultate dieser Arbeit sind für Wissenschaftler und Spezialisten interessant, die sich im Feld der Spam-Erkennung beschäftigen und das Ausnützen von Schwachstellen in der Inhaltsanalyse ihrer Spam-Filter vermeiden wollen.

# Acknowledgments

I would like to thank my advisor, Dr. Markus Strohmaier, for his dedicated guidance and support. His knowledgeable feedback to this thesis was an invaluable input. Without him and his proficiency this work would not exist.

Additionally, I want to thank my friends and colleagues at the Knowledge Technologies Institute at Graz University of Technology, who always had the right remedy in case of emergency: precious advice, coffee breaks, or funny cat pictures.

Finally, I want to thank my parents, Lilli and Karl, my sisters, Andrea and Lisa, and my fiancée Christine. Your love and support during my studies made this possible. Thank you!

I would like to dedicate this work to my grandfather, Dr. Rüdiger Axmann, who always has been a shining example of ingenuity. His ambition, creativity, and inventive genius inspired me to become an engineer.

Florian Klien
Graz, April 2013

# Contents

# List of Figures

# 1. Introduction

## 1.1. Motivation

The World Wide Web's most crucial concept is the link, which is represented by the URL[1]. It let's the user explore and browse in between web sites. When URLs become too long and unpractical for some tasks, URL Shorteners provide a service to facilitate the use of long URLs. A short-URL is the URL returned by URL Shortener services. This URL will not serve any content but instead it will redirect the visitor to the long URL. This is generally achieved with a HTTP 301[2] response. A resolve is the action of getting the long-URL from the URL Shortener by requesting the short-URL, see figure 1.1 on page 2. URL Shorteners take any URL and return a short-URL in return. The resulting short-URL is usually shorter than 30 characters. It can be used instead of the original URL. The URL Shortener now acts as a middle man between users and the world wide web. URL Shorteners redirect every request on a short-URL to the corresponding long URL. They are the link between a resource and the user. URL Shorteners are mainly used to facilitate the distribution of URLs via e-mail, SMS[3] or other channels like Twitter[4]. They may also be used to gather information on visitors of a short-link. Some URL Shorteners provide a detailed statistics page where users can track visitors of their short-links.

When the URL Shortener forwards a user to a website, they do not know where it will take them. This causes a certain responsibility on the side of the service. A URL Shortener typically wants to prevent any harm to its

[1] URL: Uniform Resource Locator
[2] HTTP 301: Moved Permanently
[3] SMS: Short Message Service
[4] `http://twitter.com`

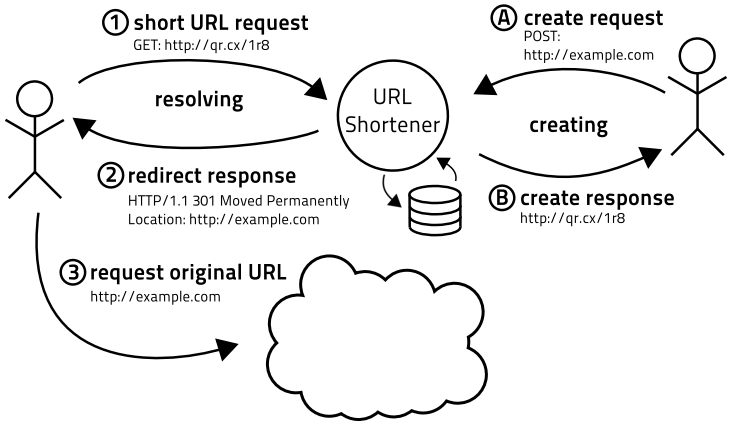



Figure 1.1.: Illustration of the basic redirect functionality of a URL Shortener. When requesting the short-link from the URL Shortener (1), it responds with a redirect to the original URL (2). The user's web-browser then takes the user to the original URL (3).
Anyone can add a URL to the URL Shortener. The creator requests a short-link (A), and the URL Shortener returns one (B).

users, which is why checking the shortened URL for malicious content is essential. Such content may include spam, scam, or otherwise fraudulent content that might harm a visitor in any way. Detecting and disabling such malicious links is paramount to keeping URL Shortening services trustworthy.

As URL Shorteners get more and more popular, it is increasingly important to detect spam or malicious short-links efficiently and effectively. Bit.ly, the largest URL Shortener, redirected users 2.1 billion times in November 2009 [09]. Although Bit.ly tries to detect spam as well as possible, their algorithms can not and do not catch every malicious short-link in their system [Mag+13].

There are some methods for spam classification that rely heavily on content. This work focuses on data that is contained within logging data or general usage data. This leads to a more efficient and less vulnerable classification approach, as content is never analyzed to identify spam. While content based approaches, which scan websites and analyze its content,

may reach a higher success rate, the difference in traffic overhead is enormous. First, the amount of data that has to be retrieved is up to the website owner. Second, content can be altered to fit spam detection algorithms but, most importantly, it can be changed at any time. A website may appear legitimate when spammers create a short-link but they can change the content later. Therefore, websites need to be rescanned in regular intervals, increasing the traffic overhead even more. Issues like congestion of popular websites, network failure, or server downtime have to be considered as well.

The approach presented in this thesis aims at minimizing traffic to external websites while simultaneously keeping the success rate of correctly identifying spam high. The objective is to reach a correct classification score of more than 95%.

## 1.2. Objectives

The objectives of this thesis are:

- to find a method for identifying malicious links in URL Shorteners
- with metrics that allow a good classification of spam by
- using as little external information as possible
- while keeping the accuracy high. The objective is to reach an accuracy of 95% or more.

## 1.3. Contribution

This thesis makes the following contributions:

- Implementation of a URL Shortening service.
- Provision of a large dataset, including click behavior in a URL Shortener.

- Presentation of an extended analysis of an URL Shortener dataset which has already been published[5]. Parts of this work have already been published in a Hyper Text 2012 short-paper by Klien and Strohmaier [KS12].
- Presentation of a set of metrics that can be used in any URL Shortener system to detect spam. These features can be easily collected and calculated to extend every URL Shortener service's current spam detection system.
- Finally, the thesis shows how the proposed metrics perform in combination with different machine learning algorithms. This insight should facilitate the planning and deployment of similar systems.

## 1.4. Thesis Outline

This work consists of five chapters. The introduction is followed by chapter 2 which gives a brief overview of the history of hypertext research, recent developments like link spamming and URL Shorteners, and finally shows the influence spam has on current on-line systems and what researchers have developed to tackle this problem.

Chapter 3 first describes the URL Shortener used to gather the used dataset. It then examines the used dataset in detail and gives an insight on how the later used metrics might help to identify spam. Chapter 4 presents and discusses the results of the classifier experiments. It includes the discussion of the experiments with its limitations. Chapter 5 concludes this thesis with an outlook.

[5] `http://qr.cx/dataset`

# 2. Related Work

This chapter provides an overview of related work to this thesis, which is not only related to spam detection but also to URL Shorteners in general and its touching points, as well as hypertext systems and web technology.

## 2.1. The Brief History of Hypertext Research

In 1945 Vannevar Bush wrote an article about the 'memex', short for 'memory index'. The machine was never built but would have been based on micro films, cameras and readers, and a mechanical system all built into one desk. The top of the desk would have allowed a user to browse documents and add comments. Bush thought of a linking structure that would allow the user to make associative links between documents. Every document could link to multiple other documents [Bus45].

Table 2.1, on page 6, shows a chronological list of the most important historic events regarding hypertext systems. The most influential event was the introduction of the World Wide Web (WWW) in 1990 by Tim Berners-Lee. He introduced a system that was not feature-complete but allowed everyone to participate. The three standards introduced with the WWW were: HTTP, the Hypertext Transfer Protocol, HTML, the Hypertext Markup Language and URL, the Uniform Resource Locator [Ber89].

HTTP allowed quick following of resources across the Internet, regardless of the resource's location. This protocol was fast and flexible. It not only allows sharing of hypertext but also of other forms of data. This versatility was one reason for its success. In combination with HTTP, the URL facilitates the identification of the resource. In addition, HTML allowed a visually more appealing representation of information. The documents could

| Year | System | Originator | Milestone |
|---|---|---|---|
| 1945 | Memex | Vannevar Bush | Microfilm-based deviceconcept |
| 1965 | Xanadu | Ted Nelson | Term 'hypertext' |
| 1967 | Hypertext Editing System | Andries van Dam, Brown University | First working hypertext system |
| 1978 | Aspen Movie Map | Andrew Lippman, MIT | First working hypermedia system |
| 1985 | Intermedia | Brown University | Anchors, webs |
| 1986 | Guide | Office Workstations Ltd. | First commercial product |
| 1987 | HyperCard | Apple Computer Corp. | Free with every Macintosh |
| 1987 | ACM Hypertext'87 | University of North Carolina | First ACM conference on hypertext |
| 1990 | WWW | Tim Berners-Lee, CERN | Hypermedia across the Internet |
| 1993 | Mosaic | NCSA | Graphical browser for WWW |
| 1994 | Hyper-G | Graz University of Technology | Hypermedia information system |

Table 2.1.: The history of hypertext systems [Mau96]

be linked to each other from within the documents. Users would follow a link embedded in the document they were reading and could be taken to another server somewhere else. Follow-up systems, like the Hyper-G system developed at Graz University of Technology, featured more complex mechanisms to include bidirectional linking, link consistency checks and many other advanced features. They came close to implementing the vision of Ted Nelson of 1965. Nelson envisioned a system that would be based on virtual inclusions. It allowed content to be included at multiple locations without being copied. As Nelson never succeeded at implementing his Xanadu system, the Hyper-G system never became as popular and widely used as the WWW. The reasons for this might be the complexity of the system, its resource management or the fact that there was never any free software available to let people implement a system of their own [Mau96].

The ability to link to any server worldwide, the ease of use, and the availability of free software led to the Internet as we know it today. A huge network of millions of servers that store petabytes of data. Or as Witten, Gori, and Numerico [WGN06] fittingly put it:

> *"The WWW is one of the greatest success stories in the history of technology. Although it exploded into the world without warning, like a supernova, the ground had been prepared over several decades: it is the culmination of the conjoint effort of philosophers, engineers, and humanities scholars. Between them, these people conjured up two revolutions, one in information dissemination and the other in human-*

*computer interaction."*

To keep up with the amount of information in a network like the Internet, there are search engines that provide the service of information filtering; information filtering in a sense as they try to match content to a query given by a user. As there is massive information on almost any topic online, search engines have to sort and prioritize content in a certain way. These metrics and methods are generally not publicly available, as they represent the business model of search engines.

## 2.2. Terminology

This section defines some essential concepts to better understand the theory behind this work.

### Uniform Resource Identifiers (URIs)

Berners-Lee, Fielding, and Masinter [BFM05] defined URIs in the following way:

> *"A Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or physical resource. [...] The URI syntax defines a grammar that is a super-set of all valid URIs, allowing an implementation to parse the common components of a URI reference without knowing the scheme-specific requirements of every possible identifier. [...] "*

Their definition later became Internet standard STD 66, [BFM05].

URI schemes should be registered with the IANA[1], but there are some schemes that are widely used without a proper registration; e.g: the 'javascript:' URI.

The name URN (Uniform Resource Name) was originally defined in 1997, but is now deprecated. Instead the more general term URI is used.

[1] Internet Assigned Numbers Authority

```
The following are two example URIs and their component parts:

      foo://example.com:8043/over/there?name=ferret#nose
      \_/   \______________/\_________/ \_________/ \__/
       |           |            |            |        |
    scheme     authority       path        query   fragment
       |   _____________________|__
      / \ /                        \
      urn:example:animal:ferret:nose
```

Figure 2.1.: RFC3986, defining the different parts of an URI [BFM05]

### Uniform Resource Locators (URLs)

Uniform Resource Locators are a subset of URIs. The main difference is that they do not define a resource by its name but by its location. The location defines a computer network or a single computer where the file can be found. The 'file' URL scheme is an example of this. 'file:///home/flo/-Desktop/diplomarbeit.pdf' accesses a PDF document in the Desktop folder in the home directory of user flo. Figure 2.1 shows URL and URI components.

The 'http' URL scheme is probably the most famous one. HTTP is the main protocol for the world wide web. The URL '`http://www.youtube.com/watch?v=06Mhn0L23Tk&hd=1`' defines the location of a resource on the web. 'www.youtube.com' defines the authority which hosts the content. 'watch' is the path of the URL, and everything behind the question mark ('?') is part of a query that will be sent to this web page. One cannot take the path or any argument to any other authority to find the same content. The only exception are authorities that are built to do exactly this. These are, for example, mirrors of web services that would fill in if the main server is overloaded or out of order.

### URL Shortener

Another example of these authorities are URL Shorteners. A URL Shortener is a web service that takes any URL from a user and provides a short-URL in return. The short-URL usually has not more than 30 characters. The short-URL can the be used as surrogate for the original long URL. Instead of going to the original server directly, a visitor will now visit the URL Shortener first. The URL Shortener will then look up the long URL and redirect the user to the original URL. This process allows the URL Shortener to collect massive data about who visits what, when and how often. Figure 1.1 on page 2 illustrates the basic URL Shortener functionality.

Most URL Shorteners would define an arbitrary name for a resource and redirect to a previously defined longer URL. Their paths would be different from the URL they link to; e.g.: '`http://qr.cx/8Ctq`' links to the same resource as the Youtube link above. Youtube's own URL Shortener is different. They reserved the Belgian domain 'youtu.be' for their service. The URL 'http://youtu.be/06MhnoL23Tk' defines the location of a video resource on their 'youtube.com' website. But part of our previously mentioned URL is found in this short-link. '06MhnoL23Tk' is Youtube's video ID and is reused to define the same resource via the main website and their URL Shortener.

### Spam

The name 'spam' originated as a fantasy name to market 'spiced ham' - SPAM, a ground pork in cans, as seen in figure 2.2 on page 10. It was well known to allied soldiers in the Second World War, where it gained a certain negative connotation for being cheap military food. 'Spam' referring to the electronic form has its name from a Monty Python sketch. The scene depicts a room full of Vikings who start to sing 'spam spam spam...' and hinder all others from continuing their conversations [WGN06]. Ironically, the word 'ham' developed over time, describing messages that are not spam.


Figure 2.2.: SPAM: The Hormel 'Spiced Ham', titular Saint of spam.
(Photo from http://flic.kr/p/bVFTpi by Tom Marshall, CC-BY-SA)

## 2.3. Recent Developments: URL Shorteners

As the short messaging service Twitter[2] became more popular, URL Shorteners emerged as a way to publish links. As Twitter did not allow messages to be longer than 140 characters, and some URLs can be significantly longer than that, posting links was a challenge. Many URL shortening services quickly evolved to allow users to shorten their long links and post the short-links on Twitter. Their presence and popularity has led to research regarding their usefulness, reliability, challenges, and dangers.

Inoue et al. [Ino+11] present a study of a URL Shortener that was used right after the Great East Japan Earthquake in March 2011. They built a URL Shortener that altered shortened links to redirect users to a CDN (Content Distribution Network). This was done to mitigate flash crowds that formed when many users visited tweeted short-links. The websites behind these links were often overwhelmed by the demand. They hosted content that was helpful after an Earthquake. The disaster's impact from

[2] http://twitter.com

the earthquake, the resulting tsunami, and the triggered nuclear accident in Fukushima Daiichi, led people to visit websites with information that would help them. As people shared links on-line in Online Social Networks (OSNs) like Twitter, many websites hosting valuable information did not withstand the number of requests. Inoue et al. [Ino+11] built a URL Shortener that could avert website overloads. The shortener transformed the submitted links to links of the CoralCDN. These coralized links are URLs altered to request the same content from CoralCDN's mirror servers. To get a coralized link, one simply adds '.nyud.net' to the domain name in the original URL, e.g.: '`http://example.com/path`' becomes '`http://example.com.nyud.net/path`' [FFM04]. The study shows that within the three days, from March 15 to March 18, their service resolved nearly 25,000 requests. Their log analysis shows that most link referrals (83.4%) come from Twitter clients or Twitter's website. Their relatively small dataset furthermore shows that most of the content they redirected to was related to disaster information, e.g.: nuke accidents, earthquakes, utility and transportation, shelters, and disasters in general. Their paper shows an interesting perspective of the use and responsibility of URL Shorteners in special circumstances, such as earthquakes [Ino+11].

Antoniades et al. [Ant+11] analyzed a dataset of short-URLs they had obtained through crawling. They crawled Twitter for short-URLs of Bit.ly and Ow.ly. Furthermore, they guessed short-URLs via brute forcing hashes, obtaining a little over 9 million links. They analyzed the lifetime, the destination, the origin, the usage location, and the popularity of short-links. Their results show that the lifetime of 50% of short-links exceeds 100 days. They notice that the countries that use their set of short-links most are the USA, Japan, and Great Britain. For some reason they do not see any usage of users from China or India in their data, which are listed among the top-5 countries regarding Internet users. Most links (>60%) are resolved in services like e-mail, instant messaging and SMS. They do not refer to any website but are only seen as a direct hit on the link. The most popular destinations for their short-links are news portals. The URL popularity describes a power-law behavior [Bre+99]. They additionally took a look at the performance of shortening services in terms of latency and shortening effectiveness. Finally they investigated the latency of short-URL services and discovered that most requests are delayed by about 0.35 seconds.

Chhabra et al. [Chh+11] did a study concerning the distribution of phishing links across Twitter. Their research analyzes the connection between users and their vulnerability to click on malicious links. They find that phishers use shortening services not to shorten their links but mostly to hide their original URL. Phishers are willing to take the risk of adding an extra hop to the victim's click path to hide their real link. The advantage of breaking most spam filters and gaining the trust of users outweighs the risk of being disconnected by the URL Shortener. Furthermore, they find that spoofing brand names is very popular amongst phishers, although they recently moved their focus from financial institutions and e-commerce sites to online social media. The most popular brands are: Facebook, PayPal, Orkut, HSBC, Habbo and Bradesco. Moreover, they look into the geographical distribution and the lifetime of phishing URLs.

Benevenuto et al. [Ben+10a] detected spammers on Twitter by finding attributes that would describe users based on their posting behavior and social connections. They use the number of hashtags per tweet, number of URLs per words, number of words of each tweet, number of hashtags per tweet, number of characters per tweet, and so on. Furthermore, they use a list of common spam words to match used words. The fraction of tweets that contain these known spam words represents another metric. The used metrics are based on observations on the user behavior of spammers, e.g.: spammers have a higher fraction of tweets containing a URL and 39% of spammers' tweets contain spam words, only 4% of the tweets by non-spam users show spam words. Also, spammers use hashtags more vigorously than normal users. In addition to the content-based metrics they also used behavioral attributes. These are for example: the number of followers, the number of followees, the age of the user account, the number of mentions, the number of times a user replied to someone, whether or not there are spam words in the user's screenname, time span between tweets, etc. In total, they used 23 attributes per user. To classify users, Benevenuto et al. [Ben+10a] used a Support Vector Machine (SVM), a state-of-the-art machine learning technique [Joa98]. The results show that they can classify spammers with 70% accuracy and non-spammers with 96% accuracy. They also show that by using only the behavioral attributes they are able to achieve a similar classification rate [Ben+10a].

Maggi et al. [Mag+13] did an extensive two-year study on short-URLs. They

analyzed almost 25 million short-URLs from up to 622 URL Shorteners. They found that the most popular shortening services blocked malicious URLs within a short period of time after creation. However, if short-URLs were created with a benign website and that content was later changed, all of those links were kept active. None of the URL-shorenters would repeatedly check the long URL for malicious content. The only exception was Tinyurl.com which deleted 1,800 spam links that became active after their short-link creation. Bit.ly allowed every malicious link to be shortened but deleted them shortly after [Mag+13].

## 2.4. Spam

Spam affects many applications. It is no longer only limited to e-mail [Cal+08] but also concerns blogs [Tho07], Online Social Networks like Twitter [Ben+10a; Gri+10], videos [Ben+10b; Ben+09; Ben+08], web search engines [FMN04], and also URL Shorteners [12b; KS12]. Therefore, researchers and engineers have developed a series of techniques to tackle this problem in different domains.

### 2.4.1. E-Mail Spam

Unsolicited e-mail or e-mail spam has been a problem for e-mail users and service providers since the early days of the ARPA net. E-mail spam is mainly used to advertise products but also to distribute viruses and malware.

There are many means to handle e-mail spam. All commercial e-mail providers offer spam filter mechanisms. There are some popular open source projects that can handle spam as well. The probably most popular open source solution is the Apache Software Foundation's SpamAssassin[3]. It is based on an implementation of several machine learning algorithms. The project first started in April 2001 and became an Apache Software Foundation project in 2004.

[3] `http://spamassassin.apache.org/`

Kanich et al. [Kan+08] conducted an experiment in infiltrating one of the largest botnets, the Storm botnet. The botnet is a massive spamming network with millions of nodes. The botnet consists of master nodes that are most likely to be controlled by the botnet owners, then there are worker bots and proxy bots. The proxy bots relay commands from the master servers to the worker bots in an encrypted P2P (peer to peer) network. The worker bots execute their orders and report back to the proxy bots. When a machine first becomes infected, it determines if it is publicly reachable. If so, it becomes a proxy bot. If it is behind a firewall, it becomes a worker. Workers can take up any task, from sending spam to participating in DDoS (Distributed Denial of Service) attacks. Kanich et al. [Kan+08] set up eight proxy bots on virtual machines. Their network traffic was tunneled through a centralized gateway to block unanticipated behavior, such as DDoS attacks. Furthermore, this gateway rewrote commands that were sent to worker bots. Doing this, they were able to let worker bots do what they wanted. Kanich et al. [Kan+08] rewrote e-mail addresses and URLs that should be included in spam messages to point to websites that they controlled. By intercepting commands from the master servers and rewriting them for the worker bots, the monitoring of the botnet's success rate was made possible. Their 'man-in-the-middle' approach allows an estimate of how much spam links are resolved by spam victims. They not only looked at click-through-rates of links but also at conversion rates of pharmaceutical online shops and executions of possibly harmful malware (Malicious Software). Preparing their experiment, they set up an online shop for pharmaceuticals and a website that encouraged users to run a program on their computer. The webshop did nothing but report back what users would have bought and what amount their purchase would have been. The spam victim could not enter credit card information but instead only got to see a 404[4] error page. The program to download and run did nothing but report back to the researcher's server that it was executed. On a real spammer's website, this would have been another instance of the botnet's software or other malware. Kanich et al. [Kan+08] find that spammers have a very low conversion rate. Out of the three spam campaigns they observed only a fraction did get delivered into user's e-mail inboxes. Just 28 purchases were triggered from 347,000,000 spam mails sent. The program

[4]404: HTML error code for 'page not found'

was sent out 123,790,966 times in two campaigns and was run by spam victims just 541 times (0.000437%). Moreover Kanich et al. [Kan+08] found that 90% of spam link visits occurred within one week of the initial spam delivery. Some are visited by crawlers, others are user-generated traffic. Just 10% of users and 30% of crawlers hit a link within the first 10 minutes. Some crawlers show a pattern of hitting links one hour after spam delivery. This suggests that some crawlers are partly configured to scan links upon delivery and others to periodically check links in spam mails [Kan+08].

Chirita, Diederich, and Nejdl [CDN05] proposed a method to identify spam based on the social connection of the inbox owner. Every e-mail address, a user has contact to, is used to build a social graph between users, which are represented by their e-mail address. This social graph is then used to analyze whether it is likely that a received e-mail message is spam or not. The connections between users are interpreted as trust votes. If a connection between users exists, the likelihood of it being spam is small. But, if the sender of the message is not to be found in the graph, this likeliness increases dramatically. Their proposed system, the Mail Rank, can be calculated locally, for every user individually, or globally, with a combined social graph for all users of the system.

Wu et al. [Wu+05] used visual features to classify spam e-mails. This step was necessary as spammers started to adapt their methods to keep their e-mails passing through newer spam filters. Spammers would integrate their message in embedded images. By analyzing the content of the images Wu et al. [Wu+05] improved their filter from 47% with only text-based filtering to 81% with image-based filtering. A combination of both even brought a 84.6% detection rate.

### 2.4.2. Spam in Hypertext Systems

To fight spam in Hypertext systems engineers need different approaches.

Castillo et al. [Cas+07] proposed a system to classify spam websites based on their connections to other websites. This network of neighbors can be rather perfidious. Their research shows that network features by themselves are not precise enough to decide whether a website is spam or not.

They needed to include content-based features to make a better classification [Cas+07].

Akismet is a plugin solution for blogs using the Wordpress[5] blogging software. It is free to use for personal blogs and requires an API key registration from its users. As of April 2011, Akismet caught 25 billion spam comments. As of March 2013, the Akismet website[6] reports to have detected over 75 billion spam comments on blogs. As Wordpress version 3 was downloaded over 65 million times and is powering over 14.7% of Alexa Internet's 'top 1 million' web sites, it prevents severe abuse of spammers [12a; 11a; 11c].

Grier et al. [Gri+10] point out that many spammers use shortener services to obfuscate their links in tweets. As URLs are not directly used in Online Social Networks, such as Twitter, it is not directly apparent what site is linked to a given shortlink. Spammers take advantage of this to hide their domains behind well-known shorteners, such as Bit.ly, Ow.ly, Tinyurl, or Is.gd. To get a short-link's destination, one has to resolve it. Spammers can complicate resolving their links by chaining multiple shorteners together. These nested URLs hamper spam detection but also increase the risk for the spammer to be deleted from either of the used shortening services. They also evaluated the use of blacklists for spam detection on Twitter. Checking their dataset against three blacklists, Google Safebrowsing, URIBL, and Joewein, they found that blacklists often lag Twitter, meaning tweets with spam links are often published before they can be blacklisted. This results in the need of re-crawling published tweets and URLs to see if the content might be spam or fraudulent content. If URLs are not found on a blacklist, they might be on it later, so rechecking the blacklists is necessary as well. Nevertheless, any redirect-URL needs to be resolved to see the final landing page that then is checked on blacklists. They also argue that certain blacklists threaten to blacklist innocent websites as they blacklist entire domains. URIBL and Joewein operate in that manner. Google Safebrowsing offers a more fine-grained approach as it blacklists URLs and not domains [10b; 12f; 10c].

[5] `http://wordpress.org`
[6] `http://akismet.com/`

### 2.4.3. Link Spam

As search engines become more and more important for finding the information one is looking for, methods to influence a search engine have been developed. The main purpose of these actions is to get one's own website listed higher in the list of results regarding a certain key word. As the exact algorithms for ranking websites are well guarded by search engines, people have to reverse engineer these metrics to optimize the position of their own website in search results. Some basic algorithms used by search engines are known from publications and let search engine optimizers estimate basic approaches that could be successful. The first search engines looked at the amount of incoming links on a web site. Page et al. [Pag+99] published the Page Rank algorithm, which is one of the reasons Google became a popular search engine. They also take outgoing links into account.

Exploiting these metrics is easy. One of the methods used is called 'link farming'. One simply puts up multiple web sites that strongly link to each other. To influence Page Rank, there are ways to optimally put links on certain nodes of the link farm, e.g. it is better to place outgoing links on pages that already have many outgoing links. Also, it is beneficial to avoid sinks, a page that has no outgoing links at all. A Page Rank given to a sink would propagate their rank to all other pages on the web uniformly. Linking to a second page inside the link farm increases the overall Page Rank of the link farm. Additionally, one needs as many external links linking to the link farm as possible. Finally, one places links on every node of the link farm to the page or community one wants to promote. Influencing other algorithms such as HITS [Kle99], which uses 'hubs' and 'authorities' as basic metric, is also possible. Hubs are pages linking to many other pages. Pages that have many incoming links are called 'authorities'. The idea is that authorities have many web sites that link to them. Ergo they have content that is worth one's while to take a look at. Hubs are more influential if they link to more prominent authorities, e.g. google.com, cnn.com, apple.com etc. Pages mentioned on many influential hubs are also called authorities. To influence HITS, one creates multiple good hubs, pages that link to many popular pages plus the one page one wants to boost. The only limit to this method is the financial threshold one can't pass to run a huge number of hubs [WGN06].


Figure 2.3.: A typical scamming website, promising easy money.

### 2.4.4. Scam, Malware and Questionable Content

Spam represents all forms of unsolicited advertisement, messages, comments or e-mails [CL98; WGN06]. Spam is ubiquitous. With a real life mailbox one would add a sticker that says "No Ads". With the electronic form of a mailbox, the e-mail provider usually does some sort of spam filtering. These are methods developed to automatically test if an e-mail is considered to be junk or spam. If it is considered to be spam, the mail server automatically delivers it to a separate folder. This way, the user never needs to hand-filter e-mails. With URLs this problem usually does not exist as people can see which link they are visiting. Before clicking on a link one can read which server will be visited. This is no longer true for short-URLs. These short-links 'hide' the true destination of the link. Users can only see where it redirects to by actually visiting the short-link. This can be exploited by spammers. They use short-links to lure visitors on web sites that users typically would not have visited willingly.

Scam is a fraudulent business scheme or swindle. Scam is similar to spam as it uses the same techniques to be distributed. However the goal of scam is very different. The scammer, the name for the person who is responsible

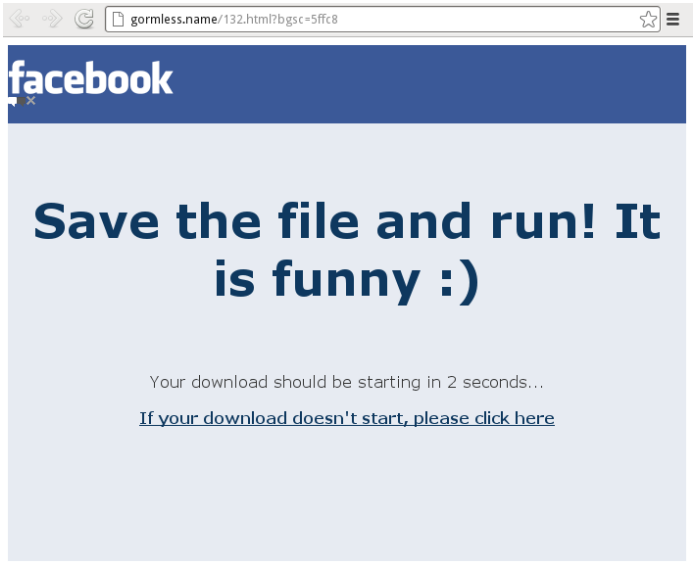



Figure 2.4.: A malware site, mimicking to be Facebook, that starts a download of a '.exe' file: 'YouLolJPG.exe'. A search of the file's md5sum revealed it to be a Trojan Horse.

for the scam, wants to financially benefit from the trustworthiness of users. Scammers will mostly promise money or something of interest and ask for something else in return. Their victim is asked to give their credit card number, bank account number, or make a down payment for a product that they will never receive. The credit information is then used to defraud the victim [And+07]. A typical example of a scam website is shown in figure 2.3. The site promises quick and easy money. The least a victim can lose on this site is their personal contact information, which may also be valuable to the scammer. Valid e-mail addresses and names are also sold on the black market [11b]. These addresses are then again used to send more spam or scam e-mails.

Another threat is malware. Malware is a type of computer program that takes advantage of the user's data or machine in any way. The most commonly known malware is the Trojan Horse. It is named after a tale in Greek mythology. The program essentially offers some kind of back door into the infiltrated computer. Unlike other forms of computer viruses, it generally

does not self reproduce. The most common use for a Trojan Horse is e.g.: Keystroke logging, screen watching, data theft or running the infected machine as part of a botnet.

Those botnets are then used for spamming or DDOS[7] attacks. Figure 2.4 shows the download site of a malware ('YouLolJPG.exe'). Its md5sum revealed that it was a Trojan Horse [12e]. The website wants to appear trustworthy and mimics the appearance of facebook.com. The real URL however shows it is not Facebook. Without the 'help' of a URL Shortener this link would not have been clicked on that easily.

Throughout this thesis I will subsume all fraudulent links, spam, scams, and malware links as just spam.

### 2.4.5. URL Blacklists

Before URL Shorteners were abused for spamming, spammers used their links for their campaigns directly. They bought a cheap domain and used the newly created URL directly. This is the reason why blacklist services like 'surlb.org' exist. [sur12]

Depending on the purpose of the malicious URLs, they might get listed in different blacklist indexes. *Wepawet* is an index for malware, *Spamhaus* lists domains found in spam e-mail, *PhishTank* lists URLs of phishing attack sites, *Google's Safe Browsing* lists URLs of both phishing and malware sites, *SURBL* lists phishing, malware and e-mail spam web sites, and *URIBL* lists domains that were used in spam e-mails, they also offer a whitelist for domains [13e; 13d; 13c; 13a; sur12; 12f].

SURBL acts as a lookup service for domains. It offers a service based on DNS[8] to save resources. It lets spam-filters query for domains that are mentioned in e-mails. The filters then can decide if they classify the message as spam or not. As spam gets detected, the content on URL-blacklists gets updated. Using DNS has several advantages over static lists. The information is more current as SURBL updates their data dozens of times per day.

[7] DDOS: distributed denial-of-service
[8] DNS: Domain Name System

The servers can be hosted by anyone and updates get pushed via the well-tested DNS eco-system, and DNS request are very efficient and fast [sur12]. An example of querying SURBL is given in listing 2.1.

Listing 2.1: "Querying 'pleksinogars.ru' manually via the UNIX host command and ping. 'pleksinogars.ru' is on the SURBL blacklist. On the other hand 'qr.cx' is not listed on the blacklist."

```
flo@rod ~ % host -tA pleksinogars.ru.multi.surbl.org
pleksinogars.ru.multi.surbl.org has address 127.0.0.4
flo@rod ~ % ping -c1 pleksinogars.ru.multi.surbl.org
PING pleksinogars.ru.multi.surbl.org (127.0.0.4) 56(84) bytes of data.
64 bytes from 127.0.0.4: icmp_seq=1 ttl=64 time=0.014 ms
--- pleksinogars.ru.multi.surbl.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.014/0.014/0.014/0.000 ms

flo@rod ~ % host -tA qr.cx.multi.surbl.org
Host qr.cx.multi.surbl.org not found: 3(NXDOMAIN)
```

As these services evolved, e-mail spammers decided to use URL Shorteners to 'hide' their links. They know that scanning e-mails is resource-intensive and that resolving and looking at the content of every link from every e-mail is too much for most spam-filters. They kick off a chicken and egg problem for URL Shorteners. When they start creating short-links for a spam-flood, no one can tell if those links should be considered spam or not. Just by looking at the URL itself, it is impossible to determine if it is spam. The URLs will not appear on blacklist as they have not been used yet. When the spam-wave hits the first servers, the spam-filters would have to resolve the short-link to see what is behind. Based on the resolved link they could make the query at a URL-blacklist but instead spam-filters start reporting that certain short-links appear in spam mails. In the best case, the URL Shortener gets abuse e-mails to let the service know about the spam-links. At this time, the URL Shortener could query URL-blacklists as well and see if the links that have been created are listed. This is however only possible after users have started using the link. If no one visits the link, the spam does not get detected.

As a result, URL Shorteners use different techniques to identify these links and disable them. *Bit.ly* and *Safe.mn*, for example, use Google's Safe Browsing or SURBL. They also use other undisclosed methods to further improve their results [12b; 12d].

# 3. Experimental Setup

## 3.1. The URL Shortener: Qr.cx

In June 2009, I started operating a URL Shortener service. The domain registered for this service was 'qr.cx'. QR standing for QR-code[1] (Quick Response Code). CX is the Top Level Domain (TLD) of the Christmas Island, a territory of Australia in the Indian Ocean. The choice for this domain was mainly influenced by its availability as the shortener should hand out QR-Codes containing the short-URL. The QR-Code allows easy and quick sharing of URLs between smartphones. Also, sharing a URL from a desktop computer to a smartphone is possible. In mid-June the shortener provided a public API[2], which led to an increased use by websites and applications. Details on the implementation can be found in appendix A.

### 3.1.1. Architecture

As big URL Shortener datasets are not publicly available, the purpose of the service was to collect data that would not be available otherwise. The main design demand was easy usage. Users should be able to shorten a link for a website while browsing on that site. This was accomplished by accepting shortening requests as path in the URL of qr.cx, see figure 2.1 on page 8 [BFM05]. E.g.: '`http://qr.cx/http://example.com`'. The idea behind this is that users can type 'qr.cx/' just before the current URL in their browsers, hit enter, and get a shortlink for that site.

[1] `http://en.wikipedia.org/wiki/QR_code`
[2] Application Programming Interface: `http://qr.cx/api.php`

Quick short-link creation is typically implemented by using a 'bookmarklet'[3]. A bookmarklet gets its name from a combination of the words 'bookmark' and 'applet'. It is basically a Java Script call that is saved as a bookmark in the browser's bookmark bar and becomes available to the user with just one click. Qr.cx also offers a bookmarklet. The downside of this approach is that it needs prior 'installation' by the user. The link needs to be dragged onto the bookmark bar to be usable. After setup, the new tool is usable by clicking on it. The main disadvantage of the bookmarklet is that it is not available on smartphones. The bookmarklet needs an environment where the 'bookmark' can be clicked on. It then reads the currently active URL in the browser window and opens the qr.cx website with the current URL as path. The called shortener website displays a short-link for the previously active website with a QR-code that can be scanned by any QR-code reader on any smartphone.

The main link creation approach, by adding 'qr.cx/' in front of the URL, allows link creation on devices that do not have the possibility of using bookmarklets, such as, for example, smart phones or tablet computers. Furthermore, it does not need any prior installation of any kind to use the qr.cx service.

To keep the service as simple as possible it does not offer user-accounts. The main reason for this was to keep the service open and available for everyone, without the need of an account or a forced sign-up. This later proved to be difficult to maintain. Without user accounts there is no mechanism to prevent abuse or spamming on a user basis. Having user accounts or API-keys would have made it much easier to block abusive users. This is still an option for future versions of the service.

The only meta-information that is available to the end-user is the click count of an URL. This can be accessed by adding a plus sign ('+') at the end of the short-URL, e.g: '`http://qr.cx/1r8+`'. The accessed information page would show the QR-Code of the short-URL as well as the original link where the short-URL would relay to.

[3] `https://en.wikipedia.org/wiki/Bookmarklet`

### 3.1.2. The Qr.cx API

The Qr.cx API is based on HTTP GET requests and allows programmers to integrate the qr.cx service into their own software. As the API sends its responses in machine-readable formats, its usage is much more convenient than implementing a service integration on top of the normal user interface.

The basic GET request for creating a short-link would be: `http://qr.cx/api/?longurl=http://example.com`. Special characters in the URL should be encoded so that they are considered URI-safe; making sure the receiver cannot misinterpret different characters in the request. A space (' ') would become '%20', '!' would become '%21', '#' would become '%23', '%' would become '%25', etc. (see percent-encoding in Berners-Lee, Fielding, and Masinter [BFM05][4]).

Apart from creating a short-URL, there is one call to query the long URL behind the short-link: `http://qr.cx/api/?get=1r8`. It returns the original URL of the `http://qr.cx/1r8` short-URL. The one optional parameter to this interface is 'mode'. It allows two values: 'plain' or 'title'. 'plain' returns the long-URL in plain text, without anything wrapped around it. The 'title' option returns an HTML anchor element with the long-URL as title, e.g.: `http://qr.cx/api/?mode=title&get=http://qr.cx/1r8` answers with:

```
<a href="http://qr.cx/1r8" title="http://example.com">
http://qr.cx/1r8
</a>
```

Any error occurring during link creation or any other request leads with 'error: ', followed by a text describing the problem. E.g.: 'error: either unsupported URL or the URL is not valid...'

The most prominent user of the API is maybe 'tiny-url.info'[5] which is a meta-URL Shortener. The service offers a common API for many URL

[4] `http://tools.ietf.org/html/rfc3986#page-12`
[5] `http://www.tiny-url.info/`

Shorteners. One basically calls an API[6] with the long URL as argument and another argument that defines which URL Shortener one wants to use.

## 3.2. The Qr.cx Dataset

The dataset, acquired by running the before mentioned URL Shortener, ranges from April 1, 2010 to December 31, 2011 and contains 732,679 short-URLs and 7,919,891 resolves of these short-URLs (see Table 3.1). A subset of this data contains geographical longitude and latitude information for users, which I obtained from geo-locating their IP. This was done by using the free GeoLite database[7]. I published the dataset under an Attribution 3.0 Creative Commons license[8] and it is available online[9]. For privacy reasons the dataset does not contain the original IP address.

Table 3.1.: Dataset characteristics: The observation period ranges from 1st of April 2010 to 31st of December 2011.

| | Clicks (Resolves) | URLs (Creates) | Sum |
|---|---|---|---|
| complete dataset | 7,919,891 | 732,679 | 8,652,570 |

There are some limitations and biases in the dataset. Given that the URL Shortener service was operated from within Austria, there likely is a local bias in the data. Furthermore, popular online social media are often hosted in US-based territories, which represents another source of bias. In addition, these services sometimes use bots to resolve short-URLs and explore the mentioned content. Other biases are possible (e.g. with regard to users' preferences with regard to certain shortener services). In general, however, one can say that - based on the comprehensive logs - the URL Shortener service reflects certain characteristics of URL Shortener services.

Figure 3.1 shows a histogram of resolves for the dataset. One can clearly see that most links were resolved less than 50 times and that very few links get

[6] http://www.tiny-url.info/open_api.html
[7] http://dev.maxmind.com/geoip/geolite
[8] http://creativecommons.org/licenses/by/3.0/
[9] http://qr.cx/dataset/

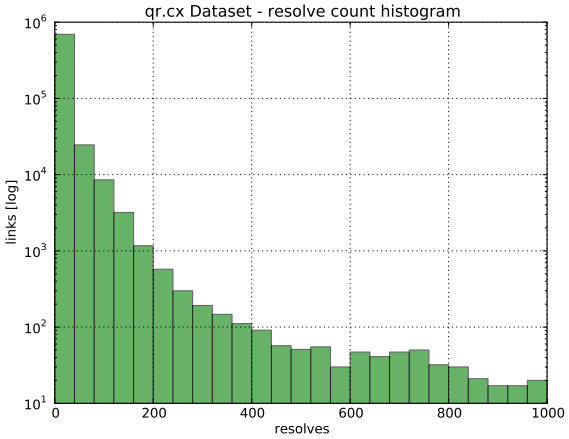



Figure 3.1.: The Resolve histogram for the dataset including resolves up to a limit of 1000. The y-axis shows a logarithmic scale count of links, the x-axis shows the number of resolves.

resolved more than 1000 times. There exist 51,675 links that have only been resolved once and one link that has been resolved around 30,000 times.

Figure 3.2 plots the number of creates and resolves over our observation period. Resolves and creates correlate strongly in the second half of 2011. The traffic for URL creates has increased by two orders of magnitude between December 2010 and December 2011. In the same time period resolves have increased by a factor of about 25. A spam wave hit the service in April 2011. This is depicted in Figure 3.2 and in a video visualization of the data that is available on-line [12c].

Figure 3.3, on page 27, depicts the number of URLs with a certain resolve count. The top left corner shows 51,675 URLs that were resolved only once. The bottom right corner shows that only three URLs have been resolved more than 10,000 times. 340,000 links, which have a resolve count of zero, are not included in the logarithmic graph.

Analyzing the resolves of links one can see interesting patterns emerge. Building edges between the country a link was created in and the country the link was resolved in, one gets a graph that shows the inter-country com-

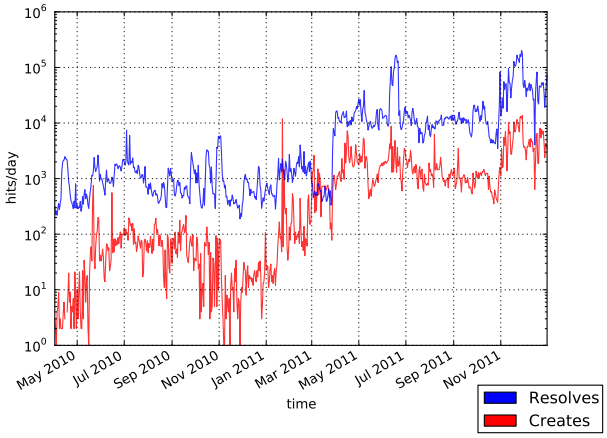



Figure 3.2.: Resolves per day are depicted in blue (upper line), creates per day are depicted in red (lower line). An increase of both creates and resolves can be observed for April 2011. Taking a deeper look into the server logs, it was found that this increase was caused by a spam wave that hit the URL Shortener service at that time.

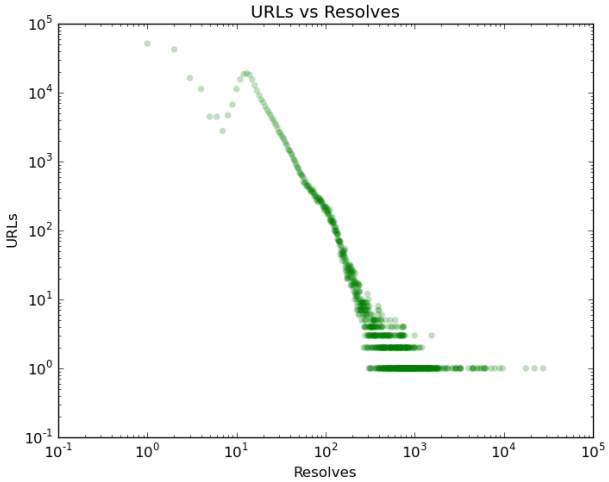



Figure 3.3.: Scatter plot of number of URLs vs Resolves. This plot depicts that 51,675 links have been resolved once, top left corner. 340,000 links, which have a resolve count of zero, are not included in the logarithmic graph.

munication that can be observed by URL Shorteners. Figure 3.4 on page 29 shows the graph of 13 countries and their resolving patterns between them. The edges' widths correspond to logarithms of their resolve count. The biggest edge counts 340,000 resolves between India and the United States of America. The weakest edge counts 5,000 resolves.

## 3.3. Descriptive Statistics

To characterize the use of the URL Shortener, a series of metrics was defined. These were used to further analyze the dataset as well as to learn from them to classify spam. The metrics introduced in section 3.3.1 are mostly meant to be used on groups of users. These groups of users can however have any scale, from simple IP subnets, to geo location-based arrangement, or parts of countries. I used IP addresses to group users by country. This allowed a look at well-defined groups of users. While this method might not equally distribute all users among all 'groups', it is a well-known pattern that is easy to understand and easy to handle.

### 3.3.1. Metrics

In order to formulate metrics some concepts within the URL Shortener network are defined. Basically, all short-links have to be created at first in order to be usable. From the moment of their creation they exist infinitely. A group of users, in this case countries, is named by their function in the network. A *creator* is the group of users that creates a short-link. The cumulative count of these actions is the 'create count' or the number of creates by that group. A *resolver* is the group of users that clicks on a short-link. This action is defined as a resolve or click. The cumulative count is the resolve count or the indegree.

As for any network, a graph is defined to consist of vertices and edges. A vertex represents a group. An edge is defined to be directed and weighted. An edge comes into existence whenever a group clicks on a short-link. The edge starts at the creator's node and ends at the resolver's node. One can

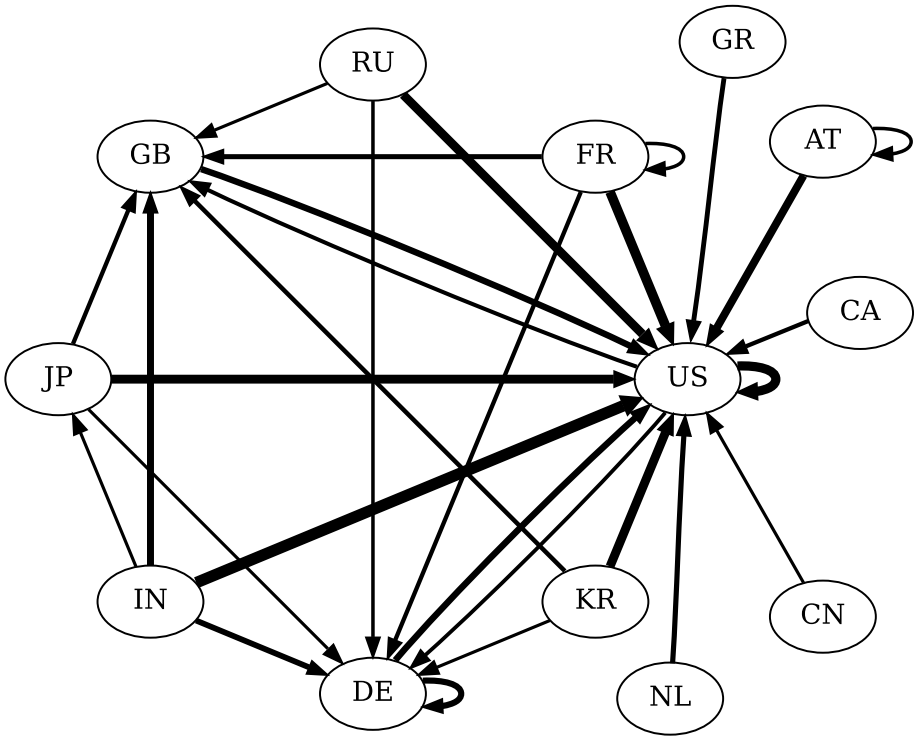



Figure 3.4.: The international flow of shortened URLs: Edges go from countries where links have been created to countries that resolve them. Edges' widths correspond to logarithms of their resolve count. Smallest edge weight is 5,000, biggest 340,000.

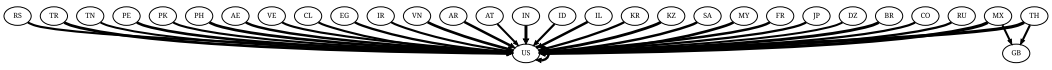



Figure 3.5.: Graph of countries showing resolves; only edges with a weight of more than 50,000 are shown. The U.S. and Great Britain are the main hubs where shortlinks are resolved.

think of it as a message that has been sent from group A to group B, from creator to resolver. The creation of a short-link does not have any influence on the graph at first. However, resolving a short-link results in the creation of an edge. Any subsequent resolve between the same actors increases the edge's weight by one. There can be edges that start and end in the same node. They are called 'loops'.

The *Indegree* is defined as the total weights of ending edges in a vertex. It is the same as the number of resolves of a group. The *Outdegree* is defined as the summarized weights of edges starting in a vertex [WF94]. The Outdegree defines how often links of one group have been clicked.

Furthermore, *Self Resolves* are defined as resolves that are performed in links that were created by the group itself. These are equivalent to the weight of a loop. *Alien Resolves* are defined as resolves of links that were not created by the resolving group. These are the Indegree of a vertex minus the Self Resolves. Having defined the basic concepts of the URL Shortener network, I can now introduce the first metric.

The *RC Ratio* is the ratio between resolves and creates. It tells us if a particular group visited more links than it created. It is a very basic estimation if the usage of the URL Shortener service is 'normal'. One would expect that most actions are resolves. If more creates are detected, one could call this an irregularity.

$$RC\ Ratio\ in\ \% = \frac{\#\ of\ Resolves}{(\#\ of\ Creates + \#\ of\ Resolves)} * 100 \tag{3.1}$$

This ratio models resolves and creates as percentage. 100% RC Ratio means the group has not created any links. If it is lower than 50% it means the group resolved fewer links than it created. This shows the group as a whole 'sends out' more links than it clicks on. One can see a plot of this metric in figure 3.6 on page 32.

A variation of this metric would be to only count resolves of self-created short-URLs, see equation 3.2. It shows how much of the created links are actually used locally.

$$Self\ RC\ Ratio\ in\ \% = \frac{\#\ of\ Self\ Resolves}{(\#\ of\ Creates + \#\ of\ Self\ Resolves)} * 100 \tag{3.2}$$

The *Internal Resolve Rate* (IRR) is the ratio of link resolves that were created by the groups themselves to its Indegree. It is expressed in percent:

$$Internal\ Resolve\ Rate\ (IRR)\ in\ \% = \frac{\#\ of\ Self\ Resolves}{\#\ of\ all\ Resolves} * 100 \tag{3.3}$$

The IRR shows how many of the resolves of a group happen on links created by that group. E.g.: If the fictional country Eurasia has 80 resolves of links that were created within the country, and 20 resolves of links that were created abroad, it would have an IRR of 80%. The links visited in Eurasia are 80% local links.

The ratios between Indegree and Outdegree show the 'consumption' or 'broadcasting' activity of a group. The Indegree is defined as the cumulative resolve count of a group. The Outdegree was defined as the cumulative resolve count of links created by that group.

The *Resolver Percentage* (Res. %) shows how much a group 'consumes'.

$$Resolver\ Percentage = \frac{Indegree}{(Indegree + Outdegree)} * 100 \tag{3.4}$$

The *Creator Percentage* (Creator %) is an indicator for the 'broadcast' activity of a group.

$$Creator\ Percentage = \frac{Outdegree}{(Indegree + Outdegree)} * 100 \tag{3.5}$$

The Resolver Percentage differs from the Internal Resolve Rate (IRR) as it does count every link resolve. Including those of links that were not created locally. Figure 3.7b, on page 34, shows the Resolver Percentage for links created in Austria. One can see that only 10% of all clicks are local. Clicks from the USA (fig. 3.7f) are mostly resolved locally. Detailed plots for most countries can be found in appendix B starting on page 66.

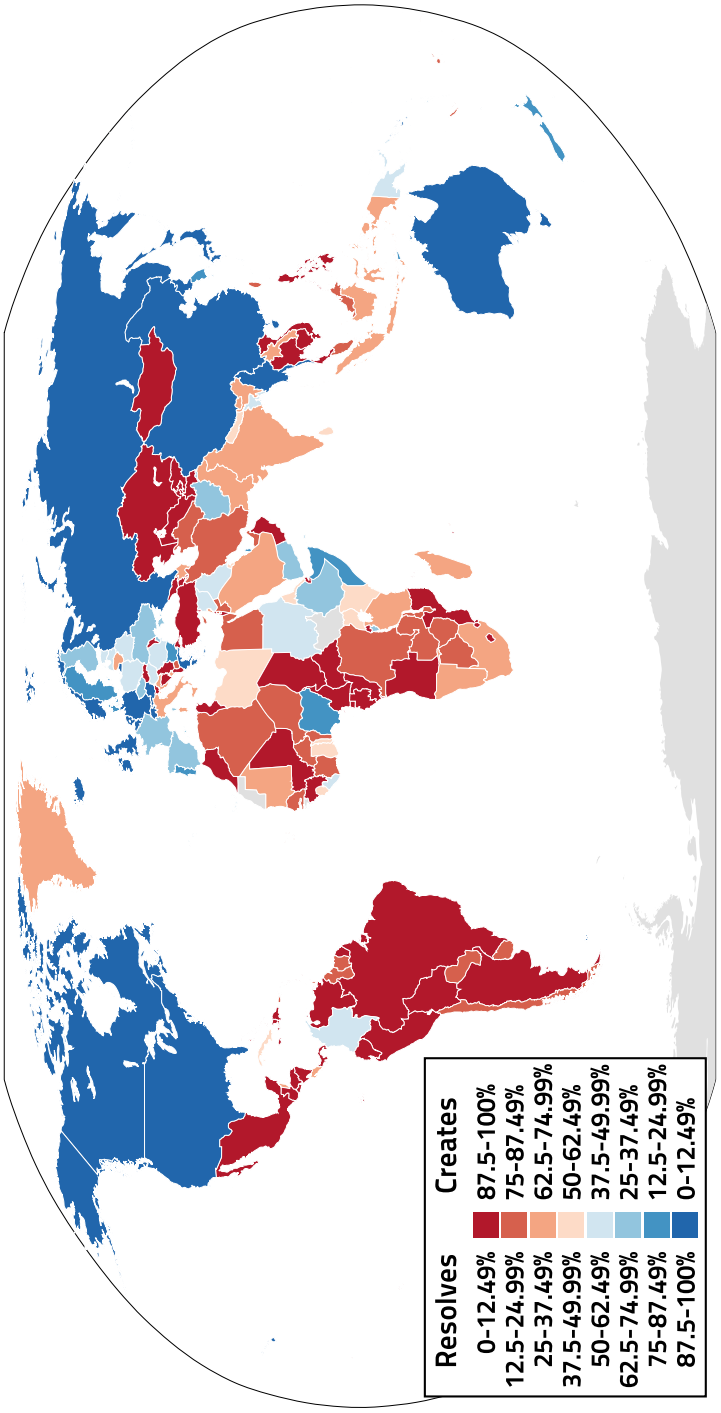



Figure 3.6.: World map showing the ratio between resolves and creates (RC Ratio) by country, as shown in equation 3.1. Large parts of South America and Africa are identified as mostly creators with small numbers of resolves. North America, Asia, Australia, and (to some extent) Europe are identified as mostly resolvers, with small numbers of creates.

Plotting the In- and Outdegree as shown in figure 3.8b on page 36, one can see which countries 'export' more links than they 'import'. A high Outdegree indicates that there are more links clicked on abroad than there are links clicked on locally. Figure 3.8a shows the world map of Resolver Percentage and Creator Percentage. Countries in red have a high Creator Percentage, countries in blue have a high Resolver Percentage.

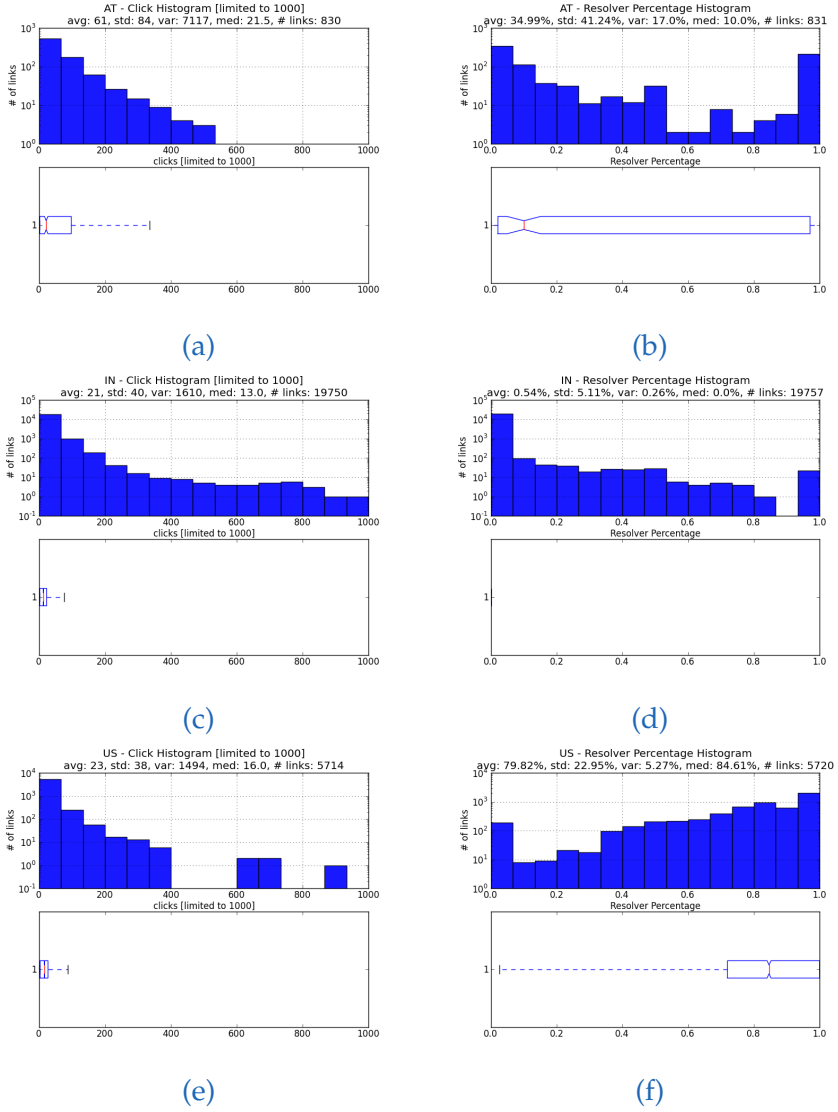



Figure 3.7.: The Click histogram for links created in different countries on the left hand side, shows the basic world wide popularity of links created in that country. The Resolver Percentage histogram for locally created links, on the right, shows how popular the locally created links are within the same country.
(a)(c)(e): Click histogram for Austria, India, and the USA.
(b)(d)(f): Resolver Percentage histogram for Austria, India, and the USA.
Most links created in India are visited abroad (d), showing a very little percentage of local resolves. Almost all links created in the USA have more than a 50% share of local resolves (f). Austria shows a high local popularity as well (b), while some links are almost exclusively resolved abroad.

In Table 3.2 on page 37 one can see the top 24 countries by Indegree. The United States of America have almost 10 times as many resolves as Great Britain. A low IRR and a high Creator Percentage are indicative for a nation to be 'spamming', although it is not conclusive evidence. One can see that the USA have an RC Ratio of 98.72 %, which shows that they are resolving a lot more links than they are creating. However, it does not tell us that there are no spam links at all coming from the USA.

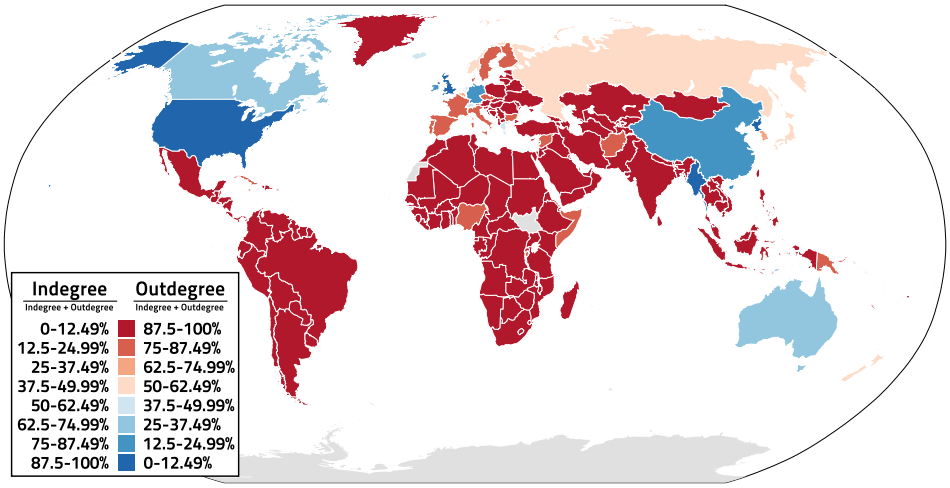
## 3.4. Annotation & Sample Sets

For the machine learning analysis of the dataset, two subsets of the big dataset, called subset A and subset B, were used. Table 3.3 on page 38 shows the amount of spam and the size of the subsets.

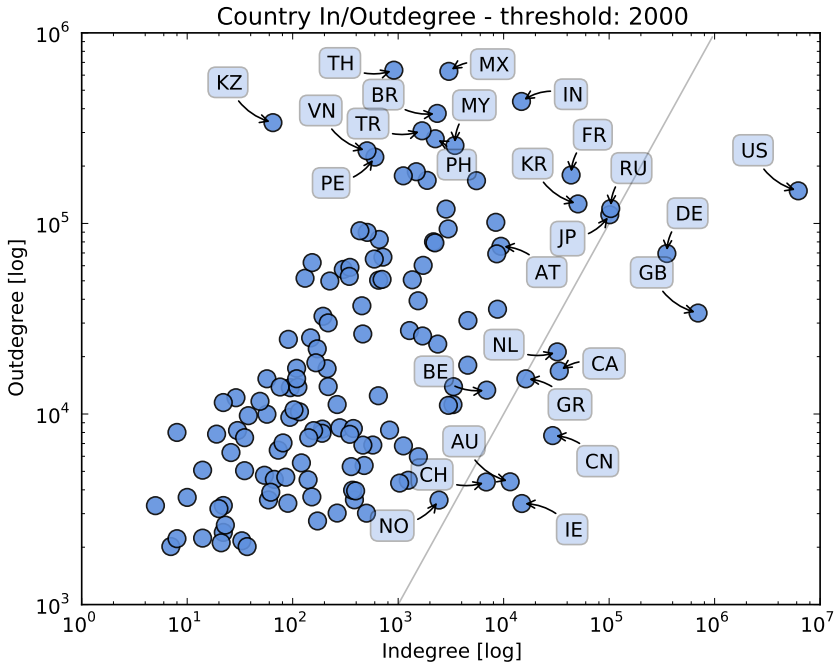
Subset A, the first and smaller sample set, contains 5,957 short-URLs, which were randomly selected and hand-annotated. A little under 1% of the original data was drawn from buckets each representing one month. Classification was done by visiting the long URL and evaluating its content. If the website was no longer available or the domain did no longer exist, it was assumed to be spam. This is based on the observation that spammers regularly discard their domains after they were blacklisted. It is likely that these domains were spam domains, but this was not checked thoroughly. If the long URL was identified as spam, all URLs that used the same domain or domain pattern were marked as spam too. This was to address a behavior of spammers that had been observed, where some spam sites would change their sub-domains or would vary their URL paths[10]. Domains which clearly host multiple legitimate websites but were abused by spammers were not handled in such a manner. The annotated set A contained 4,780 spam and 1,177 non-spam links. This results in a spam rate of 80.24%. I also evaluated whether or not creators had resolved their link themselves.

The second subset B is a semi-automatically annotated sample. It consists of 297,470 URLs. 190,883 of them are considered spam, which results in a

[10] `protocol://subdomain.domain.TLD/path`




(a) World map showing the Resolver Percentage and the Creator Percentage based on the Indegree and Outdegree of countries. Resolver Percentage is defined in equation 3.4 and the Creator Percentage is defined in equation 3.5.




(b) Scatter plot of countries by Indegree and Outdegree on a logarithmic scale. The U.S. has the highest Indegree while Mexico and Thailand have high Outdegrees. One can see which countries 'export' (on the left) more links than they 'import'.

Figure 3.8.: In- and Outdegree of Countries.

Table 3.2.: Top 24 Countries by resolves. India, ranked 12th place, shows an interesting pattern of link creates to resolves, where creates outnumber resolves twice (col. Indegree), resolves of locally created links within India are just a little over 10% of their creates. The U.S. and Great Britain show far more resolves (Indegree) than creates. Alien Resolves are resolves of links created in other countries. Self Resolves are resolves of links created within the same country. Note: **Self Resolves and Alien Resolves sum up to the Indegree.**

| Ctry | Indegree | Outdegree | Creates | Self Res. | Alien Res. | IRR | RC Ratio | Res. % | Creator % |
|---|---|---|---|---|---|---|---|---|---|
| US | **6250741** | 176437 | **81340** | 128688 | 6122053 | 2.05% | 98.72% | 96.14% | 3.86% |
| GB | **699804** | 34938 | **2409** | 2808 | 696996 | 0.40% | 99.66% | 95.24% | 4.76% |
| DE | 357036 | 70889 | 6544 | 23306 | 333730 | 6.53% | 98.20% | 83.43% | 16.57% |
| RU | 108996 | 115798 | 7599 | 3449 | 105547 | 3.16% | 93.48% | 48.49% | 51.51% |
| JP | 102979 | 113828 | 8015 | 1280 | 101699 | 1.24% | 92.78% | 47.50% | 52.50% |
| KR | 50679 | 128326 | 8965 | 3839 | 46840 | 7.57% | 84.97% | 28.31% | 71.69% |
| FR | 43886 | 213248 | 26272 | 6237 | 37649 | 14.21% | 62.55% | 17.07% | 82.93% |
| CA | 34263 | 19419 | 1154 | 4681 | 29582 | 13.66% | 96.74% | 63.83% | 36.17% |
| NL | 32454 | 20184 | 1686 | 384 | 32070 | 1.18% | 95.06% | 61.66% | 38.34% |
| CN | 30626 | 9593 | 907 | 757 | 29869 | 2.47% | 97.12% | 76.15% | 23.85% |
| GR | 16293 | 15400 | 1196 | 31 | 16262 | 0.19% | 93.16% | 51.41% | 48.59% |
| **IN** | **15620** | 456017 | **31798** | **3395** | 12225 | 21.73% | 32.94% | 3.31% | 96.69% |
| IE | 14930 | 3408 | 283 | 1 | 14929 | 0.00% | 98.14% | 81.42% | 18.58% |
| AU | 11877 | 5059 | 278 | 65 | 11812 | 0.54% | 97.71% | 70.13% | 29.87% |
| **AT** | 9580 | 77566 | 1118 | 6514 | 3066 | 67,99% | 89.55% | 10.99% | 89.01% |
| UA | 9191 | 56160 | 5208 | 158 | 9033 | 1.72% | 63.83% | 14.06% | 85.94% |
| IT | 8916 | 78610 | 16551 | 489 | 8427 | 5.48% | 35.01% | 10.19% | 89.81% |
| CO | 8449 | 115470 | 8110 | 10 | 8439 | 0.12% | 51.02% | 6.82% | 93.18% |
| BE | 6952 | 18075 | 916 | 17 | 6935 | 0.24% | 88.36% | 27.78% | 72.22% |
| CH | 6915 | 5457 | 505 | 203 | 6712 | 2.93% | 93.19% | 55.89% | 44.11% |
| SA | 5822 | 162480 | 12574 | 2405 | 3417 | 41.31% | 31.65% | 3.46% | 96.54% |
| ES | 4668 | 35624 | 2784 | 42 | 4626 | 0.90% | 62.64% | 11.59% | 88.41% |
| QA | 4601 | 18074 | 1411 | 2 | 4599 | 0.04% | 76.53% | 20.29% | 79.71% |
| MY | 3627 | 256042 | 19857 | 820 | 2807 | 22.61% | 15.44% | 1.40% | 98.60% |

Table 3.3.: The annotated subset A of the big dataset, where 80.24% of URLs are labeled as spam. The bigger semi-automatically annotated subset B consists of 64.17% spam.

| **Set A** | self resolved | not self resolved | Sum |
|---|---|---|---|
| spam | 17 | 4,763 | 4,780 |
| non-spam | 18 | 1,159 | 1,177 |
| Sum | 35 | 5,922 | **5,957** |
| **Set B** | | | |
| spam | 1,390 | 189,493 | 190,883 |
| non-spam | 837 | 105,750 | 106,587 |
| Sum | 2227 | 295,243 | **297,470** |

64.17% spam rate. The low spam rate is most likely caused by automatically whitelisting known good domains, while doing nothing with unknown domains, good or bad. The annotation of dataset B was done by creating whitelists and blacklists. Whitelisted domains would be considered safe. The whitelist contained websites like my blog, Google.com, BBC.co.uk, and many others that were shortened on a regular basis. Spam domains were added to the blacklist on a 'spam-attack'-basis. Whenever a spam-wave hit the server, those links were analyzed, the domain added to the blacklist, and the created links in the database were marked as spam. Any further links created with the same domain names were accepted but automatically marked as spam. The newly created link would never work once the domain was on the blacklist. This approach is similar to Bit.ly's behavior of accepting all URLs and then deleting malicious links later, described by Maggi et al. [Mag+13]. The resulting subsets can be seen in table 3.3.

## 3.5. Features

The main goal of my approach is to achieve a good classification score (95%+) without the need to crawl the content of the shortened website. The basic idea behind this is that, firstly, the content of a web site can change over time. This has been exploited in the past, as mentioned by Maggi et al. [Mag+13]. A spammer can easily start shortening links of websites

that host benign content at first and later change that to abuse visitors. A content check by the URL Shortener would have no impact at short-link creation time and would have to be repeated after a certain time. This has to be re-done as long as the link exists and the link is eventually blocked for abuse. This approach also leaves the spammer to use the short-link for the time between the content change and the detection by the URL Shortener. As the content of a website is no guarantee for successful spam detection it is doubtful how useful this method is.

Secondly, one has to consider network overhead for scanning websites, generating traffic, and handling server outages. This maybe less problematic than it would have been a few years ago but it still consumes resources that could be used elsewhere.

Thirdly, it is maybe not enough to just crawl the text content of websites. There are many websites that rely heavily on image content, where parts of the content is represented in images. To detect abusive content one would have to crawl those images and run character recognition on them. Clearly this is computational overhead that should be reduced as much as possible. What if one could tell if a link is spam by looking at features that can be more easily gathered? What if these features were already available to the URL Shortener? Therefore, I looked for a set of features that was able to classify spam according to my goal. This section defines the features used to classify spam and fraudulent links. The same features were used for the different machine learning algorithms.

Using the metrics from section 3.3.1 one can directly define some features to use. Those were all calculated based on the country a link was created in. The Internal Resolve Rate (*ctry_irr*), Creator Percentage (*ctry_creator_percentage*), Resolver Percentage (*ctry_resolver_percentage*), RC Ratio (*ctry_rc_ratio*), Indegree (*ctry_indegree*), and Outdegree (*ctry_outdegree*). Those were directly calculated as described in section 3.3.1. Based on their direct relation, some of these features have linear correlations. This however, did not have a negative influence on the results. The feature correlation can be seen in figure 3.9 on page 41. Table 3.4 on page 40 lists the main features.

The Create Count of a country (*ctry_create_cnt*) is the total amount of links a country has created. The Resolve Count of a country is the total number of resolves a country has performed. It is the same as the Indegree

Table 3.4.: List of features used for the machine learning experiments.

| Feature Name | Description | Comment |
|---|---|---|
| a1 - zw | up to 217 creator country features | binary feature |
| click_time | Time in minutes from creation to first click | |
| ctry | Country ID | 1 - 217 |
| ctry_create_cnt | Create count of creator country | |
| ctry_creator_percentage | Creator percentage of creator country | |
| ctry_indegree | Indegree of creator country (resolve count) | |
| ctry_irr | Internal Resolve Rate of creator country | |
| ctry_outdegree | Outdegree of creator country | |
| ctry_rc_ratio | RC Ration of creator country | |
| ctry_resolver_percentage | Resolver percentage of creator country | |
| domain_age | Domain age in minutes at link creation | |
| lat | Geographical Latitude | -180 - 180 |
| localminutes | Local day time at link creation in minutes | 0 - 1440 |
| lon | Geographical Longitude | -180 - 180 |
| minutes | Server day time at link creation in minutes | 0 - 1440 |
| numip | IP address converted to number | 0 - $2^{32}$ |
| self_click | True if creator IP visited its own link | binary feature |

(*ctry_indegree*). All links that were created within a country contribute to features with 'ctry_' in front of the feature name. These feature values are the same for any link from the same country.

The Country variable (*ctry*) is an index of the country and is arbitrarily chosen. It is basically a unique ID given to a country and is based on the temporal occurrence in the dataset.

Corresponding to the Country variable there is a set of 212 features that each represent a country. It is a binary vector. Their label is the two-letter country code (ISO 3166-1 alpha-2). Just one value out of the 212 can be set per URL, e.g.: if a link were created in Eurasia, the variable for Eurasia would be set to one and all others would be set to zero. This type of encoding overcomes certain classification difficulties with Logistic Regression.

Now I will present features that are unique to a short-URL and not based on their country of creation:

The Click Time feature (*click_time*) is the time in minutes it takes for a link to be resolved for the first time. This is zero if the link is never clicked on.

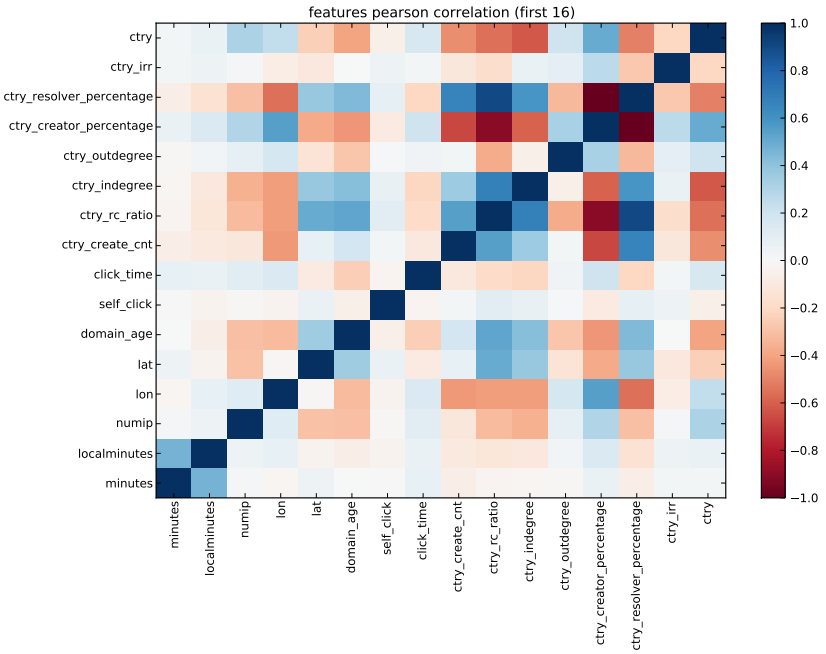



Figure 3.9.: The correlation matrix for the first 16 features (excluding the binary country features). As one would expect there is a correlation between the time of day at the creator's timezone (localminutes) and the server time (minutes). However, the time of day does not correlate with any other feature. Features based on country data correlate stronger as they are linearly dependent and derived from the same set of figures. The full correlation matrix can be seen in Appendix C.1

As the data shows, this feature is very indicative depending on whether a link is a spam link or not. Spam links have a very high Click Time.

The Self Click feature (*self_click*) is a binary feature that indicates if the creators themselves visited the link. This is not based on a group but on the single user. If the same IP created a link and is seen on the link as a user, this variable is set to one. It is reasonable to assume that spammers would not have the resources to visit every link they create. Especially as they create thousands of links at a time. A link that was created by hand has a higher chance to be checked for functionality by its creator.

The IP (*numip*) is the IP address of the creator converted into an integer. It is the base for the country information of every link as well as any other geographical information that can be used. It is the sole feature that is not available or calculable with the published dataset.

Longitude (*lon*) and Latitude (*lat*) are the geographical coordinates obtained through the IP address of the link creator.

The Minutes feature (*minutes*) is the time of day at creation based on the server time in minutes. A link created at a local timestamp of 02:00 in the morning would have a minutes value of 120.

The Local Minutes feature (*local_minutes*) is the converted local time of day based on the timezone of the creator. It is converted based on the timezone information gained from the IP address of the creator and is expressed in minutes, ranging from zero to 1440 (the number of minutes in one day). The idea behind this feature is to tell humans from programs apart. A human would be less likely to create a link at 3 a.m. than an automated program operating from another time-zone.

Finally, there is the Domain Age feature (*domain_age*). It is the only feature that cannot be directly calculated from the dataset itself. It is crawled from WHOIS records. Every domain of every link that is shortened was crawled for its WHOIS data. Most WHOIS records indicate when the domain was first registered. For this experiment this date was extracted and used as start date. The time difference between the domain's creation date and the link's creation date was calculated and converted in minutes. If WHOIS records did not indicate the first date of registration, the 'last changed' entry was used to calculate the age of the domains. If the domain did not exist any more at the time of crawling, it was assumed to be a spam domain and the Domain Age value was set to zero.

## 3.6. Spam Classification

Identifying spam is a typical classification problem with two classes: spam or ham (not spam). To classify the data four machine learning algorithms

were evaluated. Their performance is described in detail on the following pages.

The machine learning library used was the SciKit Learn Library[11] available for Python[12].

The data handling and reformatting of the data was done in Python as well. Parts of the reformatting process was implemented in the MapReduce paradigm [DG08]. This allowed significant memory savings when restructuring the data. The purpose of MapReduce is to handle huge amounts of data line by line. Furthermore, MapReduce is designed to be distributed among a cluster of machines. Every worker node does one single simple task on the one line of data it is provided with. It's called the mapper. After that, the output is directly written to the standard output. Every worker gets a subset of the data. All results are then collected and sorted. This sorted data is then aggregated by the mapper. The mapper collects all subresults and calculates a final result. A simple example for MapReduce is counting words: Mappers get a part from a document and read it line by line. For every word they see they print e.g: 'foo 1\n' to the standard output. Each line represents a key, 'foo', and a value, '1'. The sorting between mappers and reducers guarantees that all the keys are listed in one block. The reducer reads this sorted data line by line. For the same key it sums up the values. Every time the key changes it resets its counter to the current value and starts over.

### 3.6.1. Evaluation

To evaluate the results from machine learning algorithms there are several metrics one can use. As classifying spam is a classification problem with 2 classes, one can get 4 possible outputs for a classified sample: True Positive (TP), True Negative (TN), False Positive (FP), or False Negatives (FN). True Positives are correctly classified samples from the positive class, in our case spam.True Negatives are correctly classified samples from the negative class, ham, or no spam. False Positives are ham samples that wrongly got

[11] http://scikit-learn.org/
[12] http://www.python.org

Table 3.5.: Confusion Matrix: showing the four classes of binary classification.

| | | Truth | |
|---|---|---|---|
| | | Spam | Ham |
| Classifier | Spam | **TP** | FP |
| | Ham | FN | **TN** |

classified as spam. Finally, there are False Negatives which are samples of spam that were falsely classified as ham. These classes can be used to calculate different metrics:

The Accuracy, in eq. 3.6, is the ratio of samples which were correctly classified based on all samples: 'Successfully classified' versus all.

$$Accuracy = \frac{TP + TN}{(TP + TN + FN + FP)} \tag{3.6}$$

The Precision, in eq. 3.7, is the measure to evaluate how well a classification algorithm identifies the positive samples.

$$Precision = \frac{TP}{TP + FP} \tag{3.7}$$

The Recall, True Positive Rate or Sensitivity, in eq. 3.8, shows how likely it is to classify spam as spam.

$$Recall = \frac{TP}{(TP + FN)} \tag{3.8}$$

The True Negative Rate or Specificity, in eq. 3.9, evaluates how many ham samples are actually ham and not spam. Keeping this value high helps avoiding False Positives. In most spam detection systems it is better to keep this high. No ham is reported as spam and just some spam is not found. Users will find the spam anyhow but will not lose ham due to the spam filter.

$$TNR = \frac{TN}{(TN + FP)} \tag{3.9}$$

The Negative Predictive Value, in eq. 3.10. If this value is high it indicates that false negatives are low and only a little spam is missed.

$$NPV = \frac{TN}{(TN + FN)} \tag{3.10}$$

F1-Score in eq. 3.11, is the harmonic mean between precision and recall [YL99]. It has some weakness when estimating the performance of a classifier on an unbalanced dataset. As the samples in the dataset were about 80% spam, this measure was not easy to interpret. For the sake of completeness I will also report the results as F1-Score, but this value should be taken with some caution.

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{3.11}$$

$$F_1 = 2 * \frac{\frac{TP}{(TP+FP)} * \frac{TP}{(TP+FN)}}{\frac{TP}{(TP+FP)} + \frac{TP}{(TP+FN)}} \tag{3.12}$$

Matthews correlation coefficient (eq. 3.13) is a measure that works for unbalanced datasets [Bal+00]. It takes any misclassification into account. The MCC is minus one for a complete misclassification, zero for average or random performance, and one for a perfect fit. To make it comparable the MCC will also be reported in percent. The scaling was done as described in equation 3.14. 0% MCC mean no fit, 50% MCC represent a random performance and 100% MCC are a perfect fit.

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}} \tag{3.13}$$

$$MCC\% = \frac{MCC + 1}{2} * 100 \tag{3.14}$$

# 4. Results

In this chapter the results of multiple machine-learning evaluations based on those metrics and features are presented. In practical testing, four different machine-learning algorithms were used.

## 4.1. Experiments

### 4.1.1. Feature Quality

Taking a closer look at the dataset and the distribution of features in the two classes, ham and spam, one can see that many of the features look very promising to make good decisions. The latitude of a creator indicates clearly that most non spammers are located on the northern hemisphere, which can be seen in figure 4.1a. Looking at the Country Indegree (fig. 4.1h), one can see that it has about the same distribution as the Country Create Count (fig. 4.1c). This is surprising at first because when looking at the dataset one would expect that spammers come from countries with a high number of creates and a low number of resolves. This assumption is shown to be correct when looking at the Country Creator Percentage (fig. 4.1d) and the Country Resolver Percentage (fig. 4.1e), which puts the absolute values in relation to each other.

The Indegree and Outdegree from figures 4.1h and 4.1i and the RC Ratio from figure 4.1j confirm the last observation. Figure 4.1g shows that the number of different countries spammers come from is much higher than the one of non-spammers.

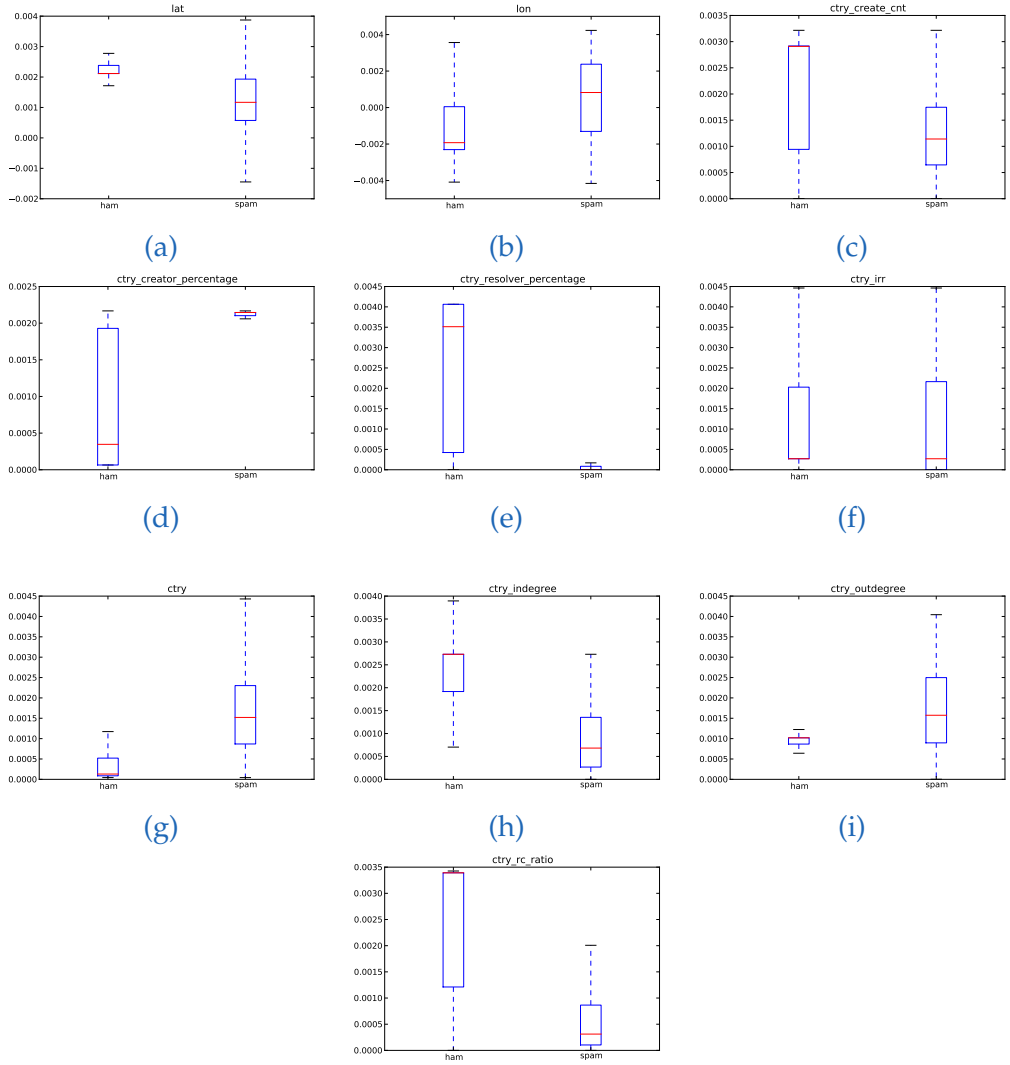



Figure 4.1.: Feature distribution differences between the two classes ham and spam.
(a): Latitude; (b): Longitude; (c): Country Create Count; (d): Country Creator Percentage; (e): Country Resolver Percentage; (f): Country Internal Resolve Rate (IRR); (g): Country ID; (h): Country Indegree; (i): Country Outdegree; (j): Country RC Ratio;
Longitude and latitude differ in their distributions across both classes (a)(b). The latitude indicates that most non-spam comes from the northern hemisphere. The creator percentage, the resolver percentage, and the outdegree show a high variance between both classes, indicating they might be valuable features.

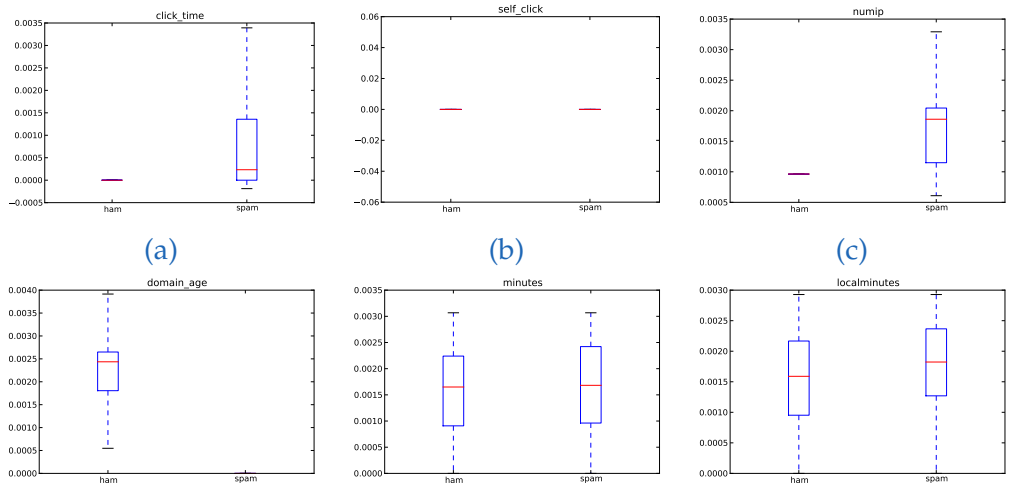



Figure 4.2.: Feature distribution differences between the two classes ham and spam.
(a): Click Time (minutes from creation to first resolve); (b): Self Click; (c): IP; (d): Domain age; (e): Minutes (link creation server time in minutes); (f): Local minutes (link creation local time in minutes);
The click time (a) indicates that spam links are idle for a much longer time than non-spam links. The day time features (e)(f) do not differ significantly between the two classes, thus signaling they might be less promising features.

The non-country related features can be seen in figure 4.2. The Click Time (fig. 4.2a), the amount of time passing between link creation and the first click, shows that spammers' links have a longer idle time before their first click. This might be due to the delay caused in a spammer's link distribution methods. The IP address of the creator has a wider range with spam than it has with ham (fig.4.2c). This is not surprising as several bot net spamming waves were observed while operating qr.cx. The time of day in minutes shows no significant difference between the two classes. Neither the server time (fig. 4.2e) nor the local time (fig. 4.2f) show significant deviance.

The most promising feature for spam detection is the age of the domain (fig. 4.2d). Spammers often change their domains. The reason for this behavior is most likely to be the existence of blacklists. The spammers' domains quickly get recognized and are then blacklisted rendering the domain unusable for their purpose. This feature is very valuable to estimate if a link is spam or not. Table 4.2 shows the results of a recursive feature elimination, which identifies it as the most important features.

Table 4.1.: This table shows the rank of the features, applying RFE (recursive feature elimination) with Logistic Regression, on subset A with 5952 URLs, rank 1 being the most important feature. Countries with a rank higher than 20 have been omitted. See appendix A Table C.1 for the full list.

| Rank | Feature |
|---|---|
| 1 | *ctry_create_cnt* |
| 2 | *ctry_outdegree* |
| 3 | *click_time* |
| 4 | *ctry_indegree* |
| 5 | *domain_age* |
| 6 | *numip* |
| 7 | *minutes* |
| 8 | *localminutes* |
| 9 | *lon* |
| 10 | *ctry* |
| 11 | *lat* |
| 12 | *ctry_creator_percentage* |
| 13 | *fr* |
| 14 | *ctry_rc_ratio* |
| 15 | *br* |
| 16 | *ctry_resolver_percentage* |
| 17 | *de* |
| 18 | *eg* |
| 19 | *it* |
| 20 | *ae* |
| ⋮ | |
| 47 | *self_click* |
| 81 | *ctry_irr* |

Table 4.2.: This table shows the rank of the features, applying RFE (recursive feature elimination) with Logistic Regression, on subset B with 297,470 URLs, rank 1 being the most important feature. Countries with a rank higher than 20 have been omitted. See appendix A table C.2 for the full list.

| Rank | Feature |
|---|---|
| 1 | *domain_age* |
| 2 | *ctry_resolver_percentage* |
| 3 | *us* |
| 4 | *ctry_rc_ratio* |
| 5 | *ctry_indegree* |
| 6 | *fr* |
| 7 | *lon* |
| 8 | *it* |
| 9 | *ctry* |
| 10 | *lat* |
| 11 | *ctry_creator_percentage* |
| 12 | *click_time* |
| 13 | *numip* |
| 14 | *mx* |
| 15 | *th* |
| 16 | *ctry_create_cnt* |
| 17 | *ua* |
| 18 | *tr* |
| 19 | *my* |
| 20 | *co* |
| ⋮ | |
| 35 | *ctry_outdegree* |
| 36 | *ctry_irr* |
| 44 | *localminutes* |
| 64 | *self_click* |
| 96 | *minutes* |

### 4.1.2. Classifier Experiments

The machine learning experiments were done with four different classifiers. The datasets used are the ones described in section 3.4. Dataset A is the smaller one with 5,957 short-URLs, dataset B is the bigger one with 297,470 short-URLs. As all classifiers ran out of memory on dataset B, only half the dataset could be used. To get any results on that subset, on a machine with 8 gigabytes of RAM, this subset had to be cut in half. The cut was done after shuffling the samples. 148,735 samples should however be enough to get a good impression of the classifiers' performances with a big dataset. The confusion matrix for the reports is based on a single training set. Two thirds of the dataset were used to train the classifier and one third was used for testing.

#### Logistic Regression

Logistic Regression's basic performance is around 85% MCC. When normalizing the data along both axes (features, samples), Logistic Regression performs even better with up to 93.7% MCC, as can be seen in table 4.4. The confusion matrix is shown in table 4.3.

#### Support Vector Classifier

Support Vector Classifiers or Support Vector Machines operate in the high dimensional space that is defined by the provided features. Their performance is highly dependent on normalized and scaled data. The results are shown in table 4.10 and the confusion matrix is shown in table 4.9.

#### Decision Trees

Decision Trees surpass the target threshold of 95% MCC. The input data for Decision Trees should not be normalized as the performance of Decision Trees would suffer. Therefore, the data was used without scaling or normalizing for this setup.

Table 4.3.: Confusion Matrix for Logistic Regression. The first confusion matrix shows the results on dataset A, the second half shows the results on a random half of dataset B.

| | | Truth | |
|---|---|---|---|
| | | Spam | Ham |
| Logistic Regression (A) | Spam | **1526** | 65 |
| | Ham | 54 | **339** |
| Logistic Regression ($\frac{1}{2}B$) | Spam | **31408** | 2477 |
| | Ham | 391 | **15303** |

Table 4.4.: Performance of Logistic Regression. ST is a simple test split with 2/3 to train and 1/3 to test, CV2 is a 2-fold cross-validation, CV10 a 10-fold cross-validation.

| **A** | ST | CV2 | CV10 |
|---|---|---|---|
| *Accuracy* | 0.9400 | 0.9378 | 0.9397 |
| *Precision* | 0.9591 | 0.9494 | 0.9553 |
| *Recall* | 0.9658 | 0.9744 | 0.9702 |
| $F_1$ Score | 0.9625 | 0.9617 | 0.9627 |
| MCC | 0.8133 | 0.7982 | 0.8064 |
| MCC% | 90.66 | 89.91 | 90.32 |
| $\frac{1}{2}$ **B** | ST | CV2 | CV10 |
| *Accuracy* | 0.9422 | 0.9421 | 0.9424 |
| *Precision* | 0.9269 | 0.9272 | 0.9276 |
| *Recall* | 0.9877 | 0.9870 | 0.9871 |
| $F_1$ Score | 0.9563 | 0.9562 | 0.9564 |
| MCC | 0.8748 | 0.8747 | 0.8755 |
| MCC% | 93.74 | 93.73 | 93.77 |

Table 4.5.: Confusion Matrix for the Support Vector Classifier. The first confusion matrix shows the results on dataset A, the second half shows the results on a random half of dataset B.

| | | Truth | |
|---|---|---|---|
| | | Spam | Ham |
| SVM (A) | Spam | **1520** | 42 |
| | Ham | 74 | **348** |
| SVM ($\frac{1}{2}B$) | Spam | **31162** | 2641 |
| | Ham | 691 | **15085** |

Table 4.6.: Performance of the Support Vector Classifier. ST is a simple test split with 2/3 to train and 1/3 to test, CV2 is a 2-fold cross-validation, CV10 a 10-fold cross-validation.

| **A** | ST | CV2 | CV10 |
|---|---|---|---|
| *Accuracy* | 0.9415 | 0.9432 | 0.9437 |
| *Precision* | 0.9731 | 0.9701 | 0.9726 |
| *Recall* | 0.9536 | 0.9587 | 0.9568 |
| $F_1$ Score | 0.9632 | 0.9644 | 0.9646 |
| MCC | 0.8215 | 0.8249 | 0.8281 |
| MCC% | 91.07 | 91.24 | 91.41 |
| $\frac{1}{2}$ **B** | ST | CV2 | CV10 |
| *Accuracy* | 0.9328 | 0.9329 | 0.9329 |
| *Precision* | 0.9219 | 0.9225 | 0.9224 |
| *Recall* | 0.9783 | 0.9774 | 0.9777 |
| $F_1$ Score | 0.9493 | 0.9492 | 0.9493 |
| MCC | 0.8533 | 0.8536 | 0.8538 |
| MCC% | 92.67 | 92.68 | 92.69 |

Table 4.7.: Confusion Matrix for Decision Trees. The first confusion matrix shows the results on dataset A, the second half shows the results on a random half of dataset B.

| | | Truth | |
|---|---|---|---|
| | | Spam | Ham |
| Decision Trees (A) | Spam | **1563** | 46 |
| | Ham | 35 | **340** |
| Decision Trees ($\frac{1}{2}B$) | Spam | **30540** | 1002 |
| | Ham | 1132 | **16905** |

Table 4.8.: Performance of Decision Trees. ST is a simple test split with 2/3 to train and 1/3 to test, CV2 is a 2-fold cross-validation, CV10 a 10-fold cross-validation.

| **A** | ST | CV2 | CV10 |
|---|---|---|---|
| *Accuracy* | 0.9592 | 0.9617 | 0.9640 |
| *Precision* | 0.9714 | 0.9777 | 0.9791 |
| *Recall* | 0.9781 | 0.9744 | 0.9761 |
| $F_1$ Score | 0.9747 | 0.9761 | 0.9775 |
| MCC | 0.8684 | 0.8802 | 0.8880 |
| MCC% | 93.42 | 94.01 | 94.40 |
| $\frac{1}{2}$ **B** | ST | CV2 | CV10 |
| *Accuracy* | 0.9570 | 0.9554 | 0.9596 |
| *Precision* | 0.9682 | 0.9662 | 0.9693 |
| *Recall* | 0.9643 | 0.9643 | 0.9677 |
| $F_1$ Score | 0.9662 | 0.9652 | 0.9685 |
| MCC | 0.9069 | 0.9031 | 0.9122 |
| MCC% | 95.34 | 95.16 | 95.61 |

Table 4.9.: Confusion Matrix for the Random Forest. The first confusion matrix shows the results on dataset A, the second half shows the results on a random half of dataset B.

| | | Truth | |
|---|---|---|---|
| | | Spam | Ham |
| Random Forest (A) | Spam | **1547** | 35 |
| | Ham | 16 | **386** |
| Random Forest ($\frac{1}{2}B$) | Spam | **31337** | 1074 |
| | Ham | 346 | **16822** |

Table 4.10.: Performance of the Random Forest. ST is a simple test split with 2/3 to train and 1/3 to test, CV2 is a 2-fold cross-validation, CV10 a 10-fold cross-validation.

| **A** | ST | CV2 | CV10 |
|---|---|---|---|
| *Accuracy* | 0.9743 | 0.9709 | 0.9740 |
| *Precision* | 0.9779 | 0.9770 | 0.9796 |
| *Recall* | 0.9898 | 0.9858 | 0.9868 |
| $F_1$ Score | 0.9838 | 0.9831 | 0.9826 |
| MCC | 0.9222 | 0.9102 | 0.9179 |
| MCC% | 96.11 | 95.51 | 95.89 |
| $\frac{1}{2}$ **B** | ST | CV2 | CV10 |
| *Accuracy* | 0.9714 | 0.9700 | 0.9715 |
| *Precision* | 0.9669 | 0.9655 | 0.9673 |
| *Recall* | 0.9891 | 0.9873 | 0.9891 |
| $F_1$ Score | 0.9778 | 0.9769 | 0.9782 |
| MCC | 0.9378 | 0.9344 | 0.9386 |
| MCC% | 96.89 | 96.72 | 96.93 |

### Random Forest

The Random Forest classifier's performance is a just about 1% MCC above the Decision Tree. The Random Forest produces about the same amount of false positives as the Decision Tree but far less false negatives, increasing the recall to 98.9%. Furthermore, compared to the Decision Trees this approach is really slow. This classifier was the slowest at learning. Its runtime performance disqualifies it from being used in a production system.

## 4.2. Discussion

Looking at the results from the classifier experiments, one can clearly see a difference in performance for each of the four methods.

Logistic Regression has an MCC of more than 90% depending on the amount of data available. The accuracy of 94.2% is quite satisfying but the True Negative Rate of 86% shows that there is a high number of false positives, which would result in numerous good links being classified as spam. The recall of 98.7% is only surpassed by random forests. This shows that very little spam links are missed by this method.

Support Vector Machines achieve a 93.2% accuracy which is almost as good as the previous method. Yet, once more the True Negative Rate of 85.1% is too low. The classifier would falsely classify too many benign links as malicious. With a recall of 97.7% the number of undetected spam links would remain very low.

The Decision Tree has an accuracy of 95.9%. The MCC surpasses 95% at all tests with the random half of subset B. There are just 1002 misclassified benign links, resulting in a True Negative Rate of 94.4%, which is the best result of all classifiers. Although the recall of 96.7% puts the Decision Tree at the last place, the precision is at 96.9%, which is the best result of all four classifiers.

The Random Forest classifier proved to have the highest accuracy. With 97.1% accuracy it is the best classification method. It also succeeds to surpass the MCC of the Decision Tree with 96.9%. With a recall of 98.9% it has the highest rate of all classifiers to detect spam. Just 346 spam links were not found when training and testing on a random half of subset B. This results in a negative predictive value of 97.9%. A True Negative Rate of 93.9% just misses the mark of the Decision Tree by 0.5%.

Overall, the Random Forest, therefore, provides the best performance of all classifiers.

## 4.3. Limitations

This experiment was performed by using a dataset of a rather small URL Shortener. The performance of the presented methods on other datasets is unknown, as there is no similar dataset available. The data certainly has a bias regarding the use of this shortener, and it is currently unknown how the same features and classifiers would perform on another dataset. The dataset at hand was significantly influenced by spammers. They were using the service regularly within the observation period and used bot-nets to post their links. The geographical distribution of bot-net nodes is one of the reasons as to why these features work so well on this dataset. Another bias of the dataset could be data points originating from Austria. As I am from Austria, the URL Shortener found its first users in Austria. The local social network has a certain influence on how the shortener was used in the beginning.

Using the proposed methods in a production system would certainly require tweaking the machine learning algorithms. The results in the previous sections are based on default settings of those algorithms. Although several settings were tried to get the best results out of every classifier, the final results are based on their default settings within SciKit-learn. The only alteration between the classifiers was the normalization and scaling of the data, as some of them need specially formatted data to function correctly.

# 5. Conclusion

The goal of this work was to show that it is possible to detect spam links within a URL Shortener network using a content-independent approach. It can be done by identifying features that do not involve any content of the shortened website. Just taking behavioral data of users, geographical data and domain information is enough to classify spam with a 95% accuracy. Excluding the content of a shortened website has multiple advantages. It saves resources as there is no need to crawl content, reformat, index and analyze it. Furthermore, the content of a website can change over time, which means that content crawling would have to be done over and over again. Crawling shortened websites can be omitted using the presented approach.

Building this classifier was achieved by identifying 16 features that vary significantly between the two target classes: spam and non-spam or ham. These features are mainly property features of users of the URL Shortener. First, the users are grouped by country. Then, an international country graph is generated which shows the inter-country traffic produced by the short-links. The properties from this graph are used to deduce information about links originating from different countries. Further features such as the domain age, the longitude and latitude are taken into account. Finally, behavioral features such as the time between link creation and the first click are used to train classifiers. These classifiers then separate malicious links from benign content. It was shown that Decision Trees perform best for this problem.

Further this work contributes a URL Shortener dataset which has already been published[1]. This dataset was collected by implementing a URL Shortener and operating it for 21 months. For privacy reasons the original IPs

[1] `http://qr.cx/dataset`

were removed from the published dataset. It contains more than 700,000 short-URLs and a little under eight million clicks.

## 5.1. Implications

The results of this work show that by taking advantage of implicit connections within a dataset one can derive useful information. This information is often self-contained and does not need to be expanded by adding additional external data. A thorough analysis of the available data points can give deep insights into connections that reveal hidden information.

By using the proposed techniques to identify spam on a URL Shortener network, operators could significantly reduce their spam detecting cycles and therefore improve their spam detection rate. They can build a completely self-reliant spam detection system. This methods can not be as easily influenced as starting a website with benign content and changing it later to become malicious. These methods have already been exploited to get spam links into URL Shorteners. The best approach would be to use the proposed method as a supplement to existing blacklists. As URL Shorteners are often the first to reveal new malicious links, this approach could detect the links that are not yet known to blacklists.

Attacks against the proposed techniques could be done by simulating 'normal' usage on spam links. This could be done by creating short-links from countries that are not peculiar in their link creation statistics. Further the link usage has to resemble the usage of a non-spam link. Tricking the spam detection system into classifying a link as non-spam is possible. However, the portion of the visitors that are exploited has to be significantly lower in order to keep the link usage at inconspicuous levels. This kind of manipulation can only be done with access to many hosts and IP addresses. Using the same machines of a botnet for creating links and faking usage might, however, give the manipulation away. The overhead for faking normal usage is likely to be too much effort for spammers. The most difficult feature to exploit is probably the domain age. It is not cheap to keep many domains for a long time. The amount of domains spammers register and discard is high. Once domains are on a blacklist they become unusable for

spammers and their purpose. Keeping one domain for a long time and then using it in just one spam attack in order to lose it just right afterwards is not a cheap approach.

## 5.2. Outlook

This work should be seen as a start point for further research in this direction. The presented features could be used to train different classifiers. One could likely use differently preprocessed data to achieve a higher accuracy. The geographical data inherent in this dataset is alluring to try clustering algorithms on it. Furthermore, there exist many more features that can be inferred from the very same dataset. Such features could be: detailed data regarding the visits on short-links, additional data from the inter-country graph, more meta-data from WHOIS-records, or meta-data from top level domains, such as domain prices or registrars.

As this dataset is available online[2], I invite the corresponding research community to improve on the methods presented in this work.

[2] `http://qr.cx/dataset`, (CC-BY)

# Appendix

# Appendix A.

# Implementation

## A.1. Version 1.0

The main programming language used is PHP[1]. The reasons for choosing PHP are the following: It is one of the most popular, if not the most popular, scripting language to generate dynamic websites. It is used by Wikipedia[2], Facebook[3], Wordpress[4], and many others. It is easy to learn, which makes it easy to find developers. It is supported by most webhosters and therefore easy to scale or migrate. PHP is object-oriented which makes inclusion of external libraries and modular development a lot easier. Furthermore, there are ways to compile PHP code to make it faster. One method to do so was developed by Facebook [10a; 13b]. Finally it provides a big and active community which is helpful when encountering any kind of difficulty.

The web-service uses an Apache HTTP webserver[5] which was configured with the mod_rewrite module[6] to allow the special treatment of URLs as paths. This could have been directly implemented in PHP. However, for quick prototyping purposes this was avoided. This step was implemented in PHP with version 2.0 of the implementation.

[1] http://www.php.net
[2] http://wikipedia.org
[3] http://facebook.com
[4] http://wordpress.org
[5] http://httpd.apache.org
[6] http://httpd.apache.org/docs/current/mod/mod_rewrite.html

MySQL is used as database backend. The database consists of just two tables. One to save the URLs and one to record the visits. The URL-table consists of: *urlid*, to save a unique ID; *shorty*, the unique random string that is used in the short-URL, e.g.: 1r8; *url*, the most important field, to store the long-URL to which the user will be redirected; *creationdate*, a timestamp of the date and time of creation; *creatorip*, to save the creators IP address; and three binary fields to store the state on deleted, approved, or hidden. A URL would be marked 'deleted' if it was regarded to be spam or would link to illegal content. It would be marked 'approved' if it was on the whitelist. The 'hidden' field was meant to be used to hide URLs from the public statistics page. It should also be available for registered users to hide their links on demand. This feature will be released at some point in the future.

The second table, which stores the visits, is used to log every visit of every short-link. The fields consist of: a unique *hitid*, giving every visit a unique ID; *urlid* referring to a URL from the first table; *date*, the date and time of the visit; *ip*, the IP of the visitor, and *referrer* the link referee sent with the browser in case a short-link was clicked on on a website. The table structure can be seen in figure A.1.

## A.2. Version 2.0

The re-implementation of the service which was done in 2011 did not go live. This implementation was done as part of a lecture called 'Multimediale Informationssysteme 2'.It is build on Symfony 2[7], a very popular PHP 5 framework. The backend used was MongoDB[8].

The main advantage of the re-implementation was the Model View Controller (MVC) paradigm. It separates code from the document structure in web-pages and allows easier restructuring of websites without touching the code behind it.

[7] `http://www.symfony.com`
[8] `http://www.mongodb.org/`

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| urlid | int(11) | NO | PRI | NULL | auto_increment |
| shorty | varchar(40) | NO | UNI | NULL | |
| url | varchar(3500) | NO | | NULL | |
| creationdate | timestamp | NO | | CURRENT_TIMESTAMP | |
| creatorip | varchar(16) | NO | | 0.0.0.0 | |
| deleted | tinyint(1) | NO | | 0 | |
| approved | tinyint(1) | NO | | 0 | |
| viewable | tinyint(1) | NO | | 1 | |

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| hitid | int(11) | NO | PRI | NULL | auto_increment |
| urlid | int(11) | NO | | NULL | |
| date | datetime | NO | | NULL | |
| ip | varchar(20) | NO | | NULL | |
| referrer | varchar(4000) | YES | | NULL | |

Figure A.1.: Structure of database tables. The upper table is used to store short-URLs and their corresponding long-URL. The lower table is used to log link-usage.

Views are based on Twig[9], a template engine for PHP. Twig allows easy iterating through lists without the need of a single line of PHP code. This way, one can list dynamic elements without calling PHP code directly. This is solely done by passing arguments. Twig is developed by Fabien Potencier, who also is responsible for the Symfony framework.

The basic idea of version 2 was to use the same gateway for users of the website and users of the API. The main qr.cx website would be just another implementation of the service's API. Using Java Script and JSON (Java Script Object Notation) the main website accesses the API as any other external software would. The API calls available to the very own web service would be available to everyone else. The most obvious advantage was the consolidation of two code bases to one. Checking for erroneous URLs, spam checking, whitelisting, and link usage logging would all be done by the backend.

To limit abuse from external API users an API-key would have to be passed

[9] http://twig.sensiolabs.org/

with the other arguments. As this API-key would be visible in the Java Script implementation on the main website, abusive users could copy this key. To guarantee that the main website could always access the API, and others that could not use the main API-key, an IP whitelisting was set in place.

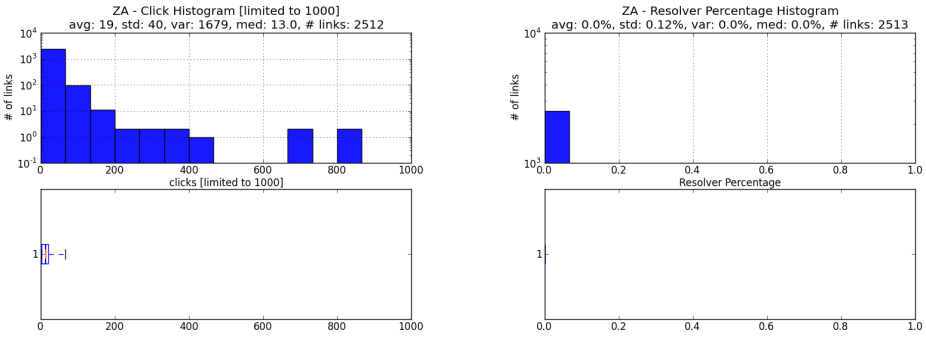
There were plans to add additional features to the API. One would be granted access to the click history of links. This click data would not only give the total number of clicks a short-link received but also the time and date of the clicks. The main website would access this API interface and would get a JSON object containing an array with certain click statistics. These would then be plotted using a library called 'jqPlot'[10]. JqPlot allows dynamic visualizing of graphs that are completely rendered in Java Script. Using this approach, one lessens the bandwidth as well as the computational resource load on the server side.

[10] `http://www.jqplot.com/`

# Appendix B.

# Histogram Country Plots

This section lists Click histograms and Resolver Percentage histograms for different countries. The Click histogram shows the short-link click count distribution on links created within the given country. The Resolver Percentage histogram shows the percentage of clicks, locally created short-links get, from within the own country. The left side has links that get no clicks from within the own country, the right side has links that get all clicks from within the creation country.




(a) (b)

Figure B.1.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side. (a): Click histogram for South Africa. (b): Resolver Percentage histogram for South Africa.

# Appendix B. Histogram Country Plots

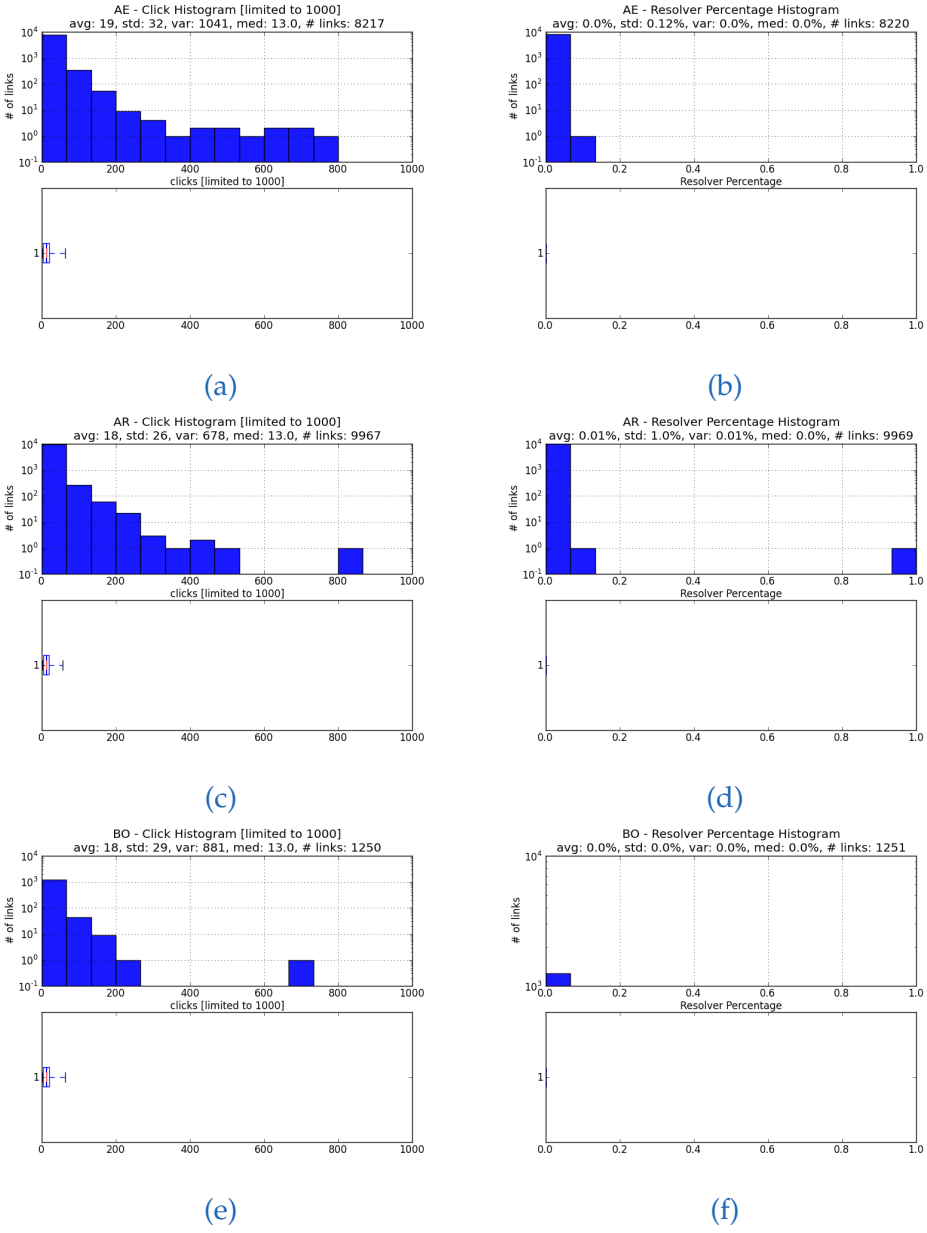



Figure B.2.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side. (a): Click histogram for the United Arab Emirates. (b): Resolver Percentage histogram for the United Arab Emirates. (c): Click histogram for Argentina. (d): Resolver Percentage histogram for Argentina. (e): Click histogram for Bolivia. (f): Resolver Percentage histogram for Bolivia.

# Appendix B. Histogram Country Plots

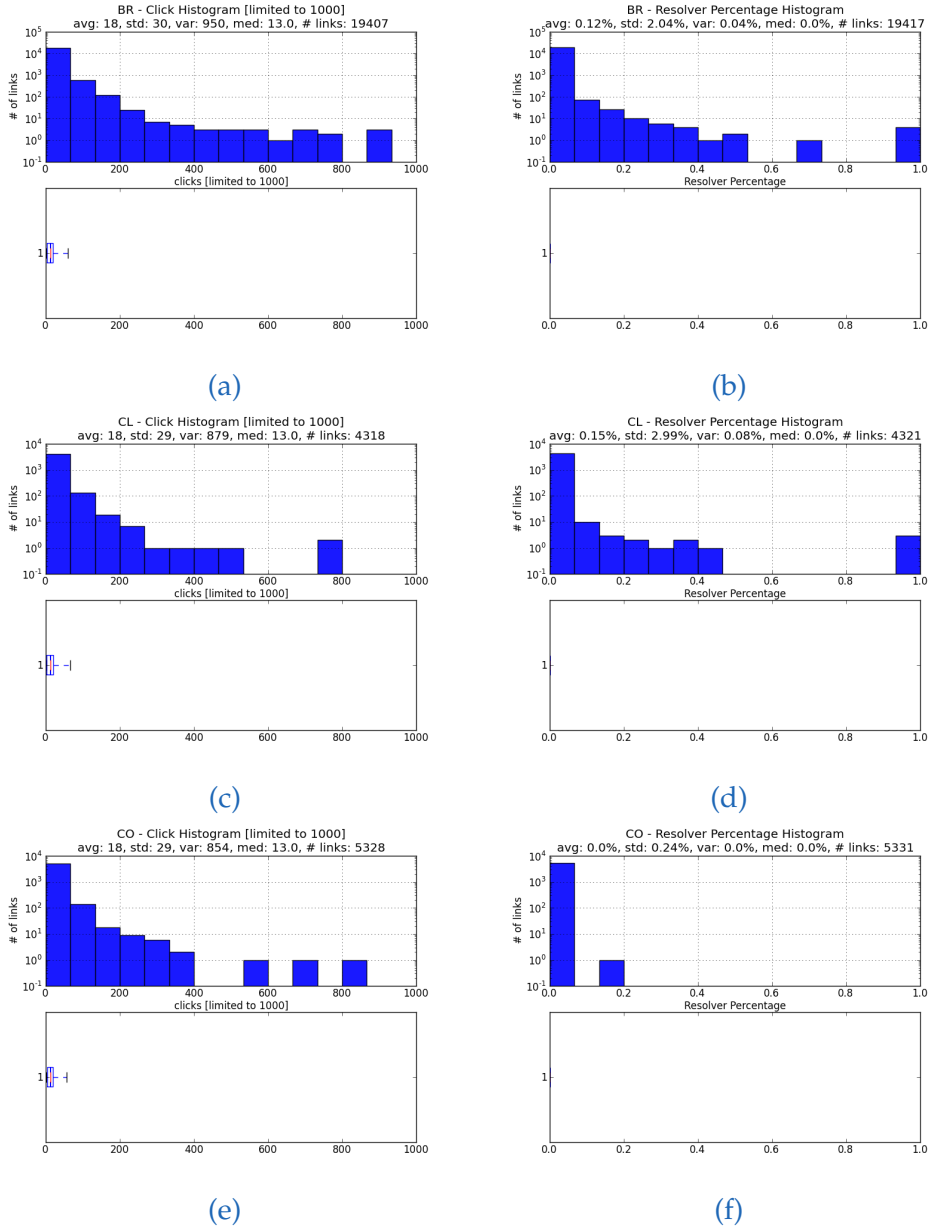



Figure B.3.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side. (a): Click histogram for Brazil. (b): Resolver Percentage histogram for Brazil. (c): Click histogram for Chile. (d): Resolver Percentage histogram for Chile. (e): Click histogram for Colombia. (f): Resolver Percentage histogram for Colombia.

# Appendix B. Histogram Country Plots

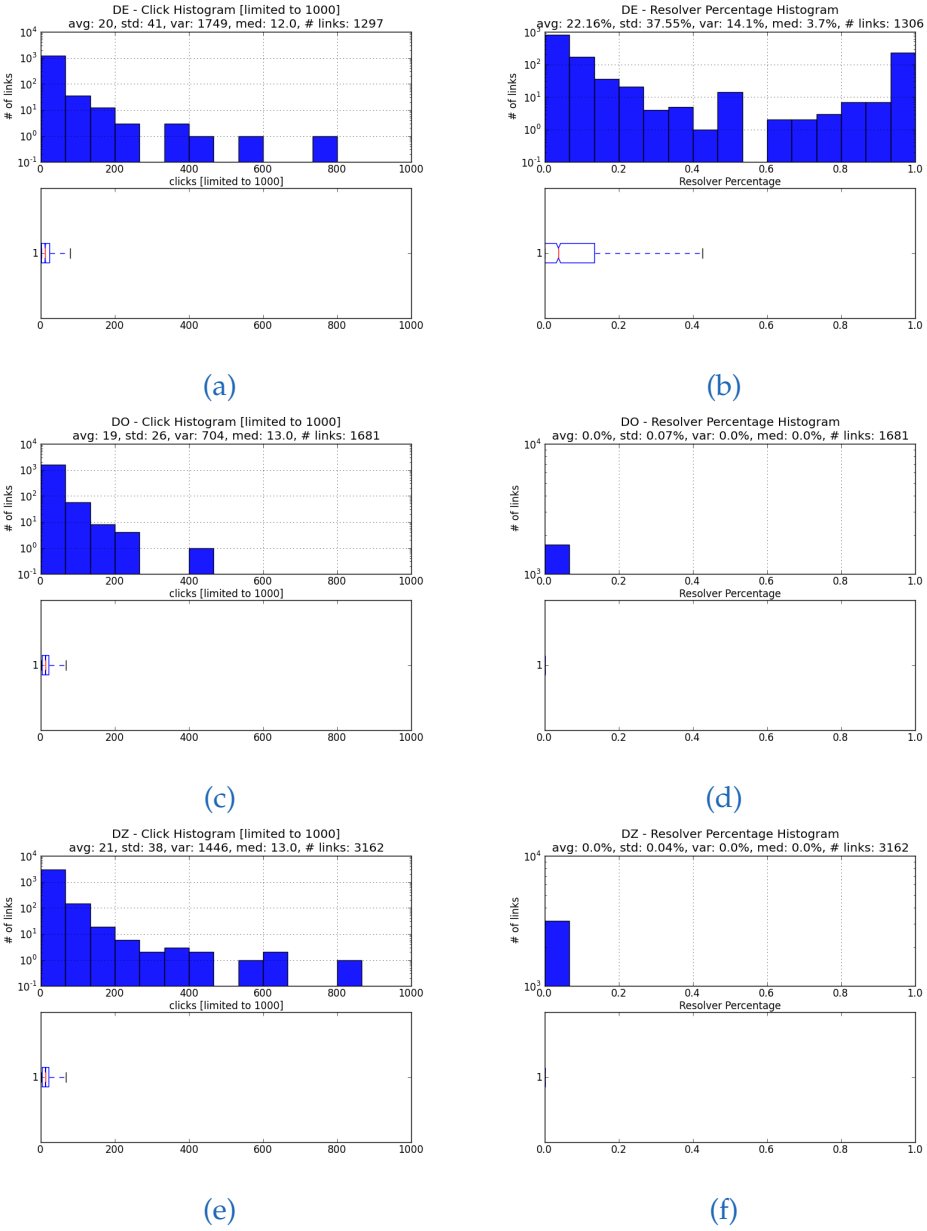



Figure B.4.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side. (a): Click histogram for Germany. (b): Resolver Percentage histogram for Germany. (c): Click histogram for Dominica. (d): Resolver Percentage histogram for Dominica. (e): Click histogram for Algeria. (f): Resolver Percentage histogram for Algeria.

# Appendix B. Histogram Country Plots

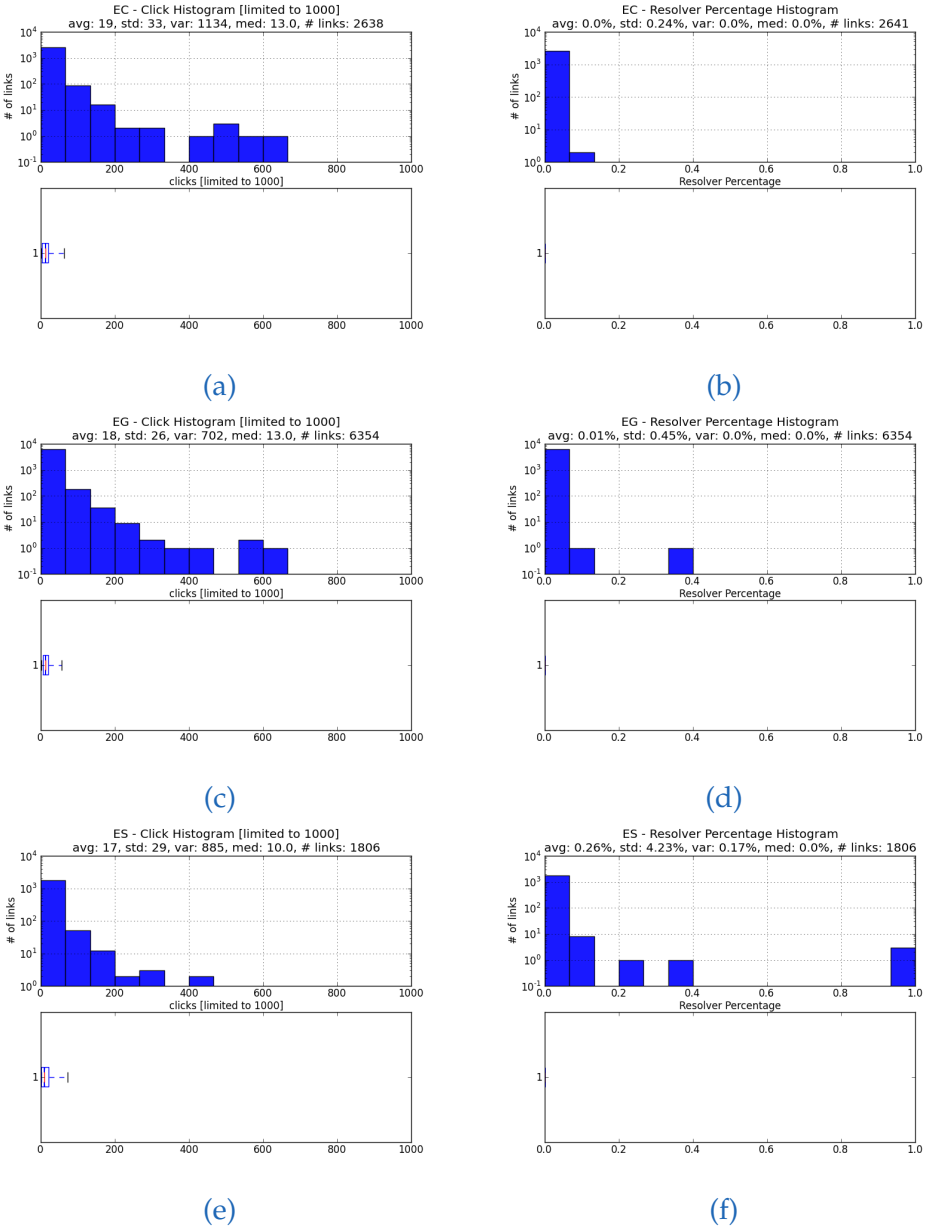
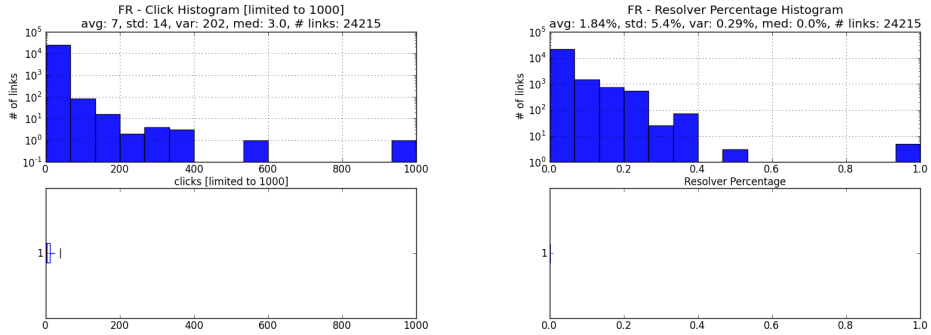



Figure B.5.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side. (a): Click histogram for Ecuador. (b): Resolver Percentage histogram for Ecuador. (c): Click histogram for Egypt. (d): Resolver Percentage histogram for Egypt. (e): Click histogram for Spain. (f): Resolver Percentage histogram for Spain.

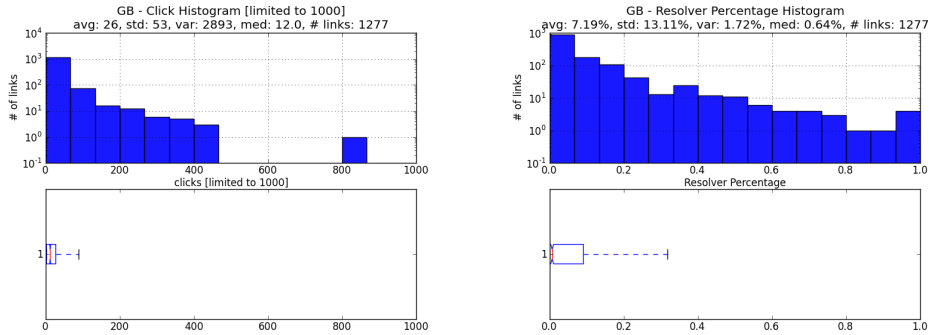
# Appendix B. Histogram Country Plots




(a)



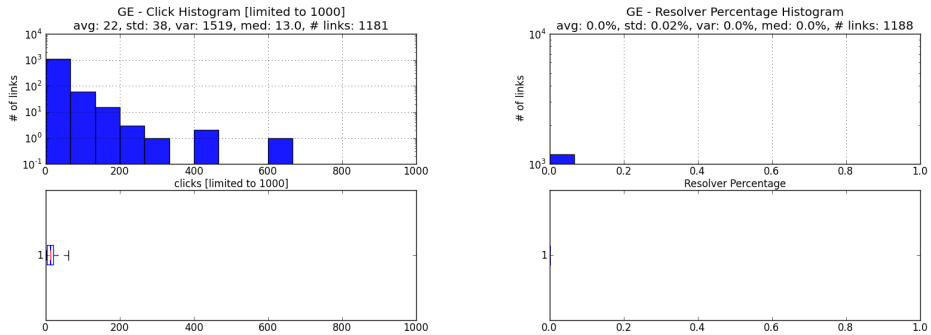

(b)




(c)




(d)




(e)




(f)

Figure B.6.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side. (a): Click histogram for France. (b): Resolver Percentage histogram for France. (c): Click histogram for the United Kingdom. (d): Resolver Percentage histogram for the United Kingdom. (e): Click histogram for Georgia. (f): Resolver Percentage histogram for Georgia.

# Appendix B. Histogram Country Plots

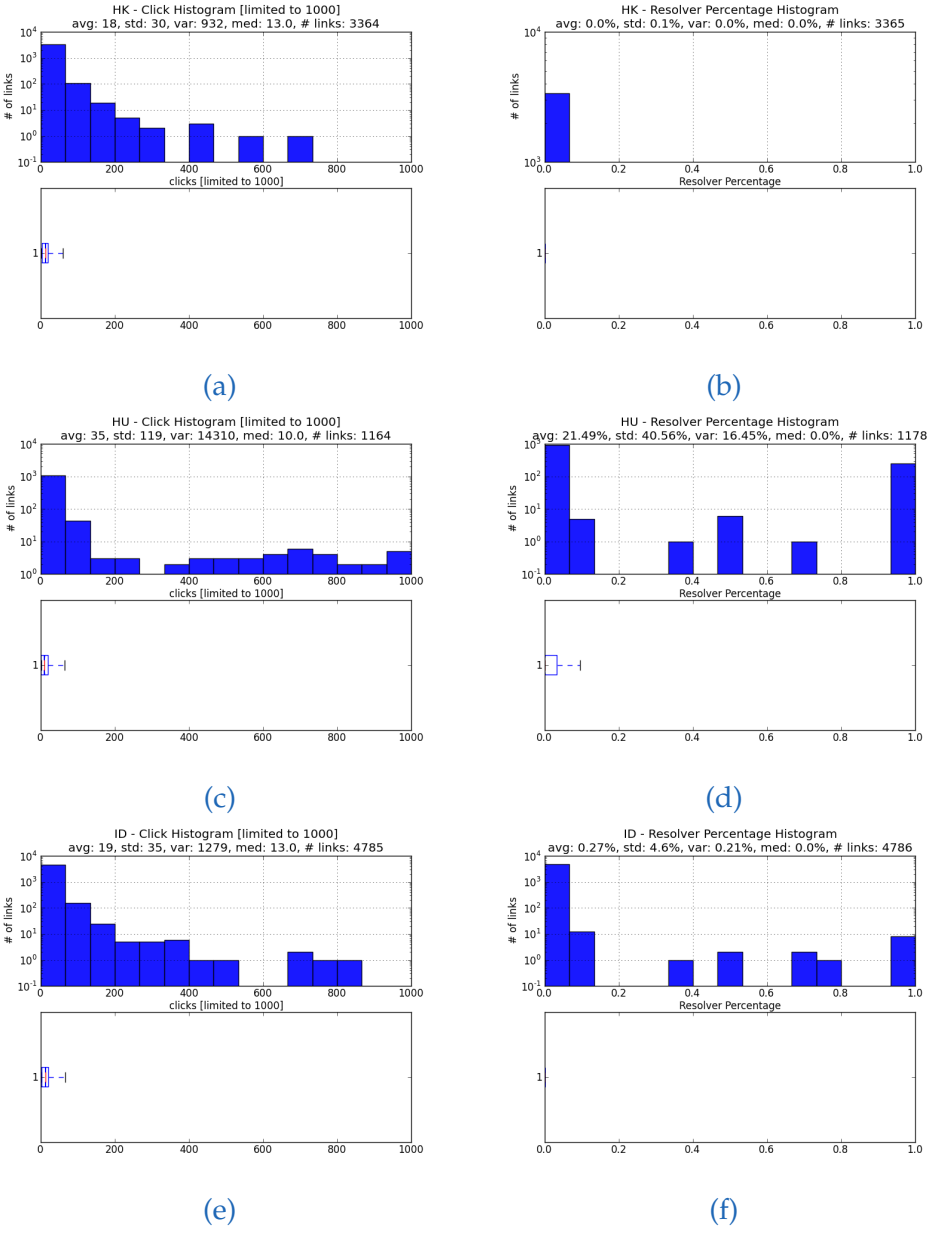
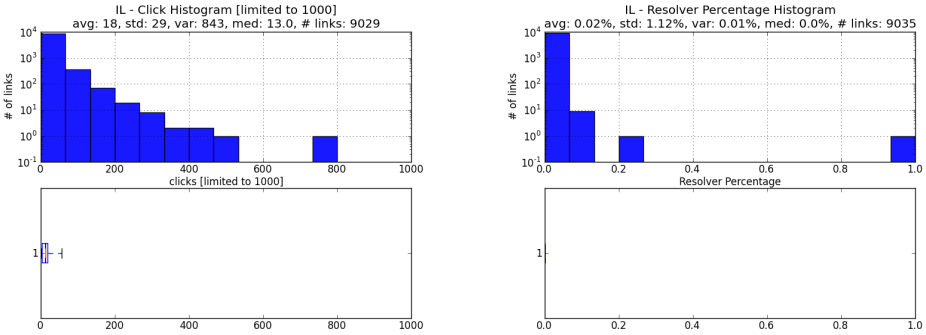



Figure B.7.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side. (a): Click histogram for Hong Kong. (b): Resolver Percentage histogram for Hong Kong. (c): Click histogram for Hungary. (d): Resolver Percentage histogram for Hungary. (e): Click histogram for Indonesia. (f): Resolver Percentage histogram for Indonesia.

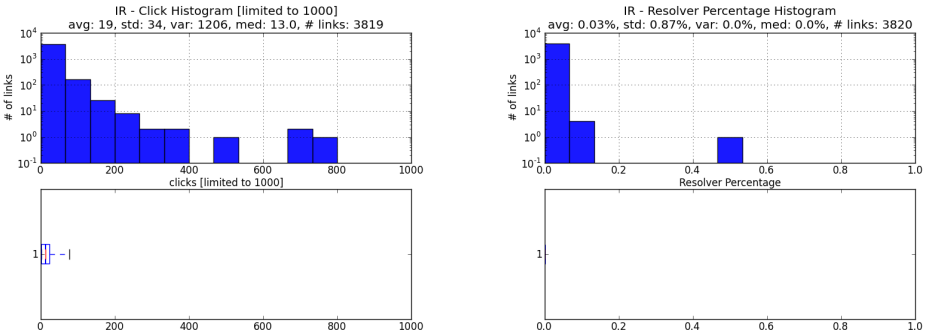
# Appendix B. Histogram Country Plots




(a)



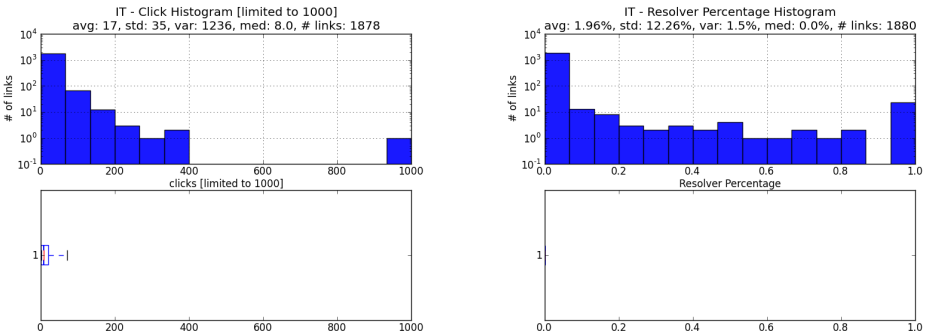

(b)




(c)




(d)




(e)




(f)

Figure B.8.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side. (a): Click histogram for Israel. (b): Resolver Percentage histogram for Israel. (c): Click histogram for the Iran. (d): Resolver Percentage histogram for Iran. (e): Click histogram for Italy. (f): Resolver Percentage histogram for Italy.

# Appendix B. Histogram Country Plots

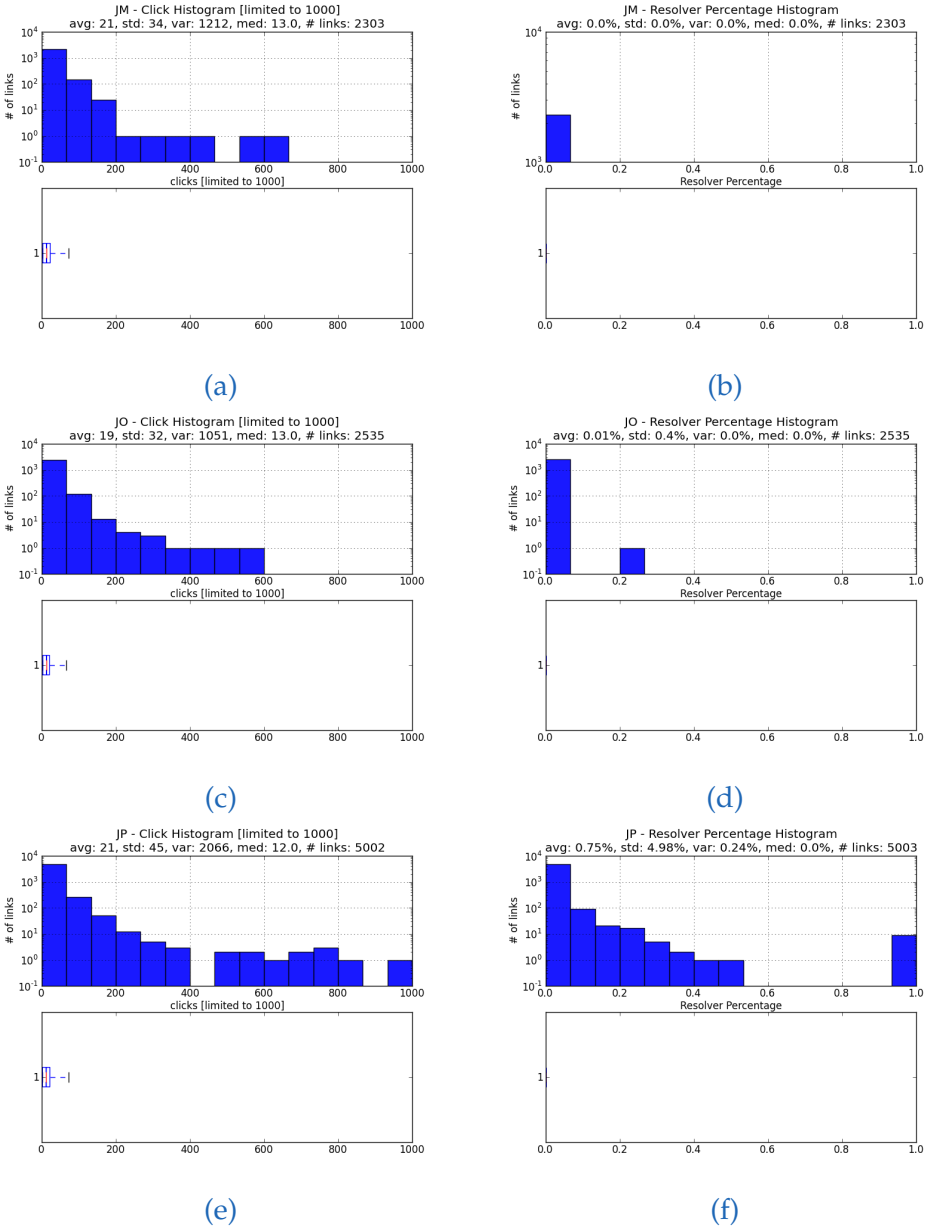
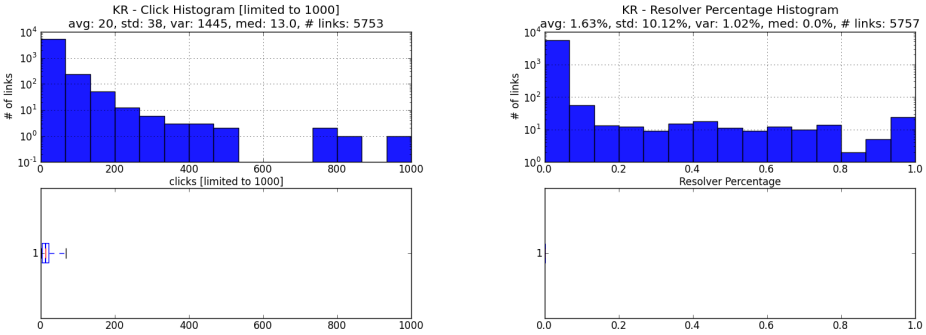



Figure B.9.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side. (a): Click histogram for Jamaica. (b): Resolver Percentage histogram for Jamaica. (c): Click histogram for Jordan. (d): Resolver Percentage histogram for Jordan. (e): Click histogram for Japan. (f): Resolver Percentage histogram for Japan.

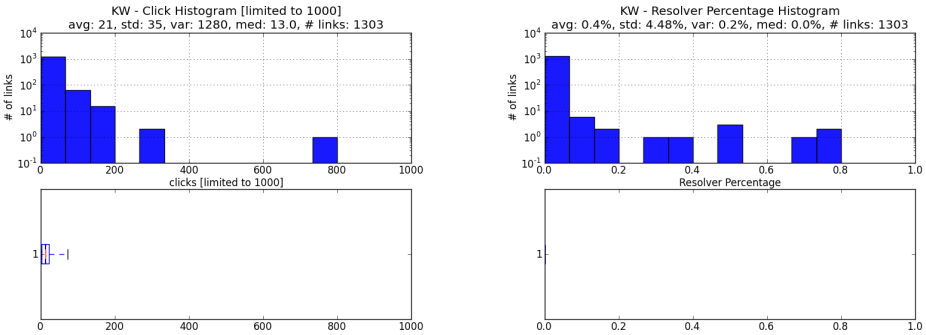
# Appendix B. Histogram Country Plots




(a)



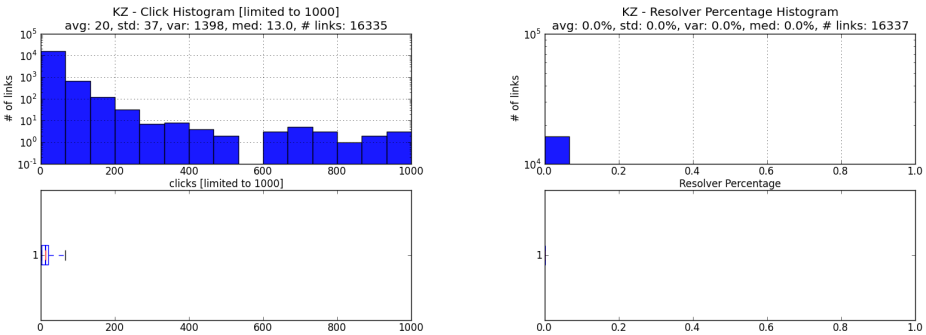

(b)




(c)




(d)



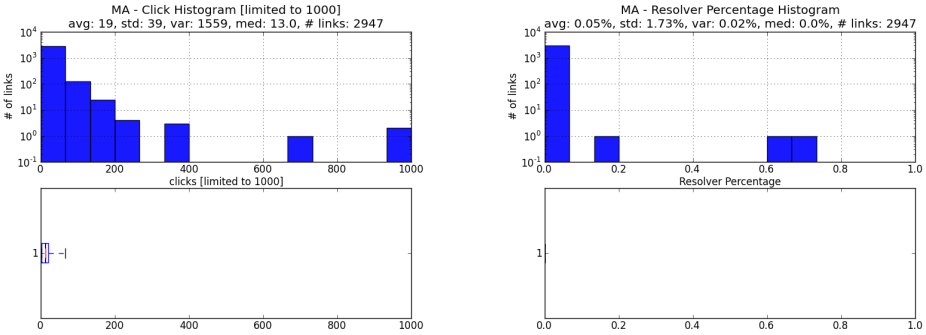

(e)




(f)

Figure B.10.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side.
(a): Click histogram for South Korea. (b): Resolver Percentage histogram for South Korea. (c): Click histogram for Kuwait. (d): Resolver Percentage histogram for Kuwait. (e): Click histogram for Kazakhstan. (f): Resolver Percentage histogram for Kazakhstan.

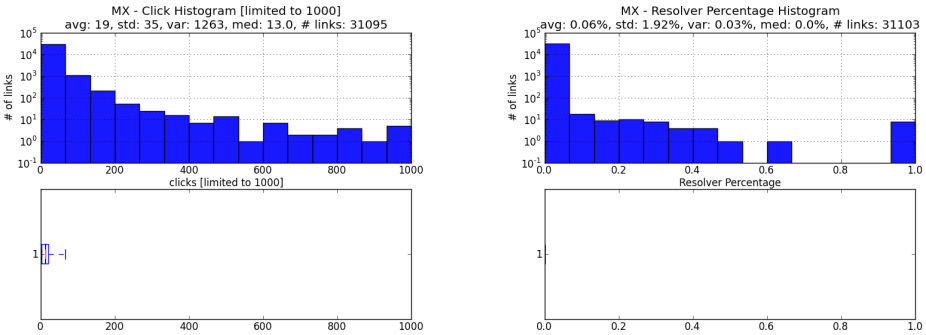
# Appendix B. Histogram Country Plots




(a)



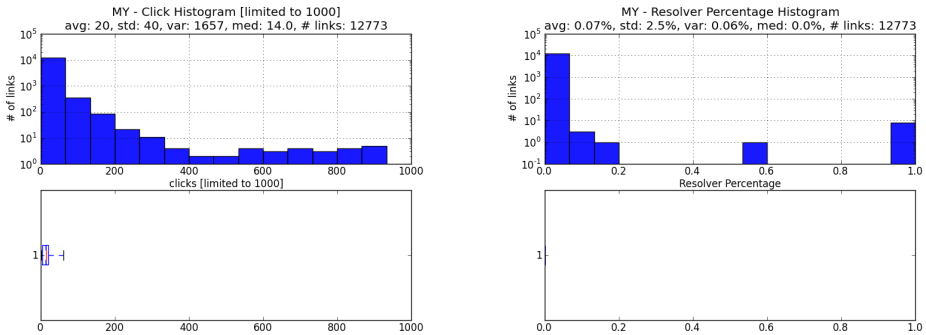

(b)




(c)




(d)



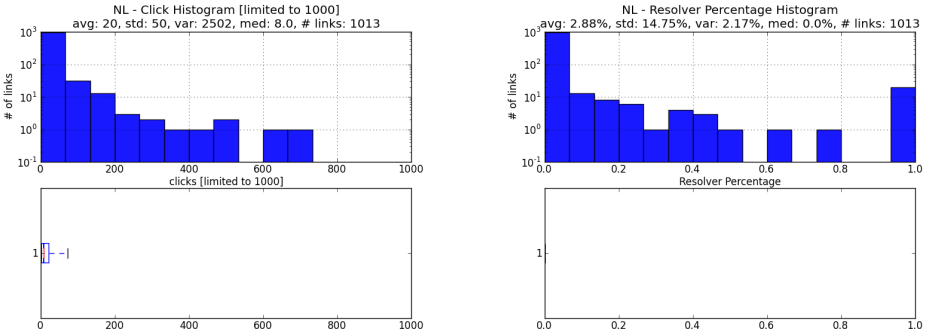

(e)




(f)

Figure B.11.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side.
(a): Click histogram for Morocco. (b): Resolver Percentage histogram for Morocco. (c): Click histogram for Mexico. (d): Resolver Percentage histogram for Mexico. (e): Click histogram for Malaysia. (f): Resolver Percentage histogram for Malaysia.

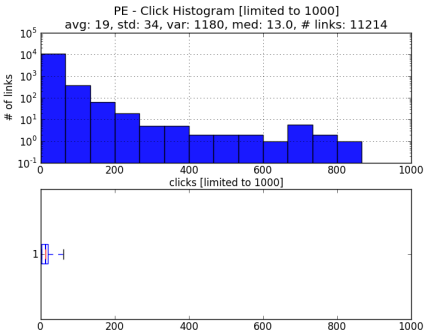
# Appendix B. Histogram Country Plots



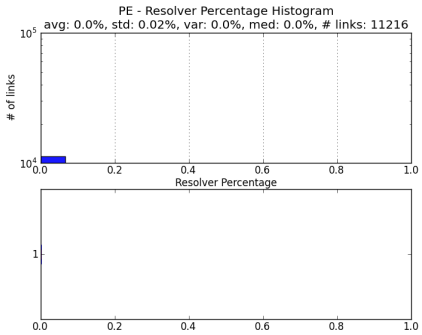

(a)



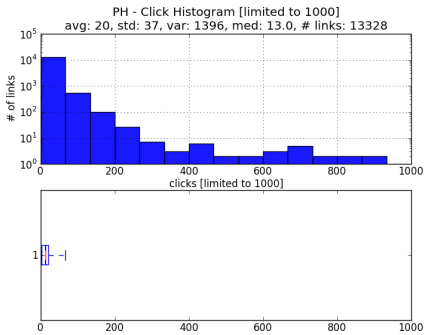

(b)



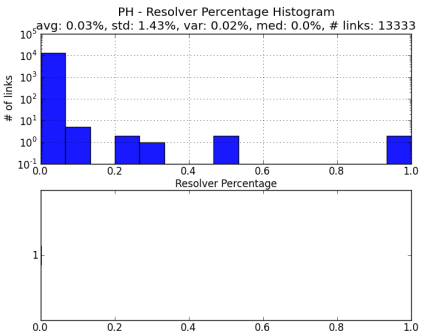

(c)




(d)




(e)




(f)

Figure B.12.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side.
(a): Click histogram for the Netherlands. (b): Resolver Percentage histogram for the Netherlands. (c): Click histogram for Peru. (d): Resolver Percentage histogram for Peru. (e): Click histogram for the Philippines. (f): Resolver Percentage histogram for the Philippines.

# Appendix B. Histogram Country Plots

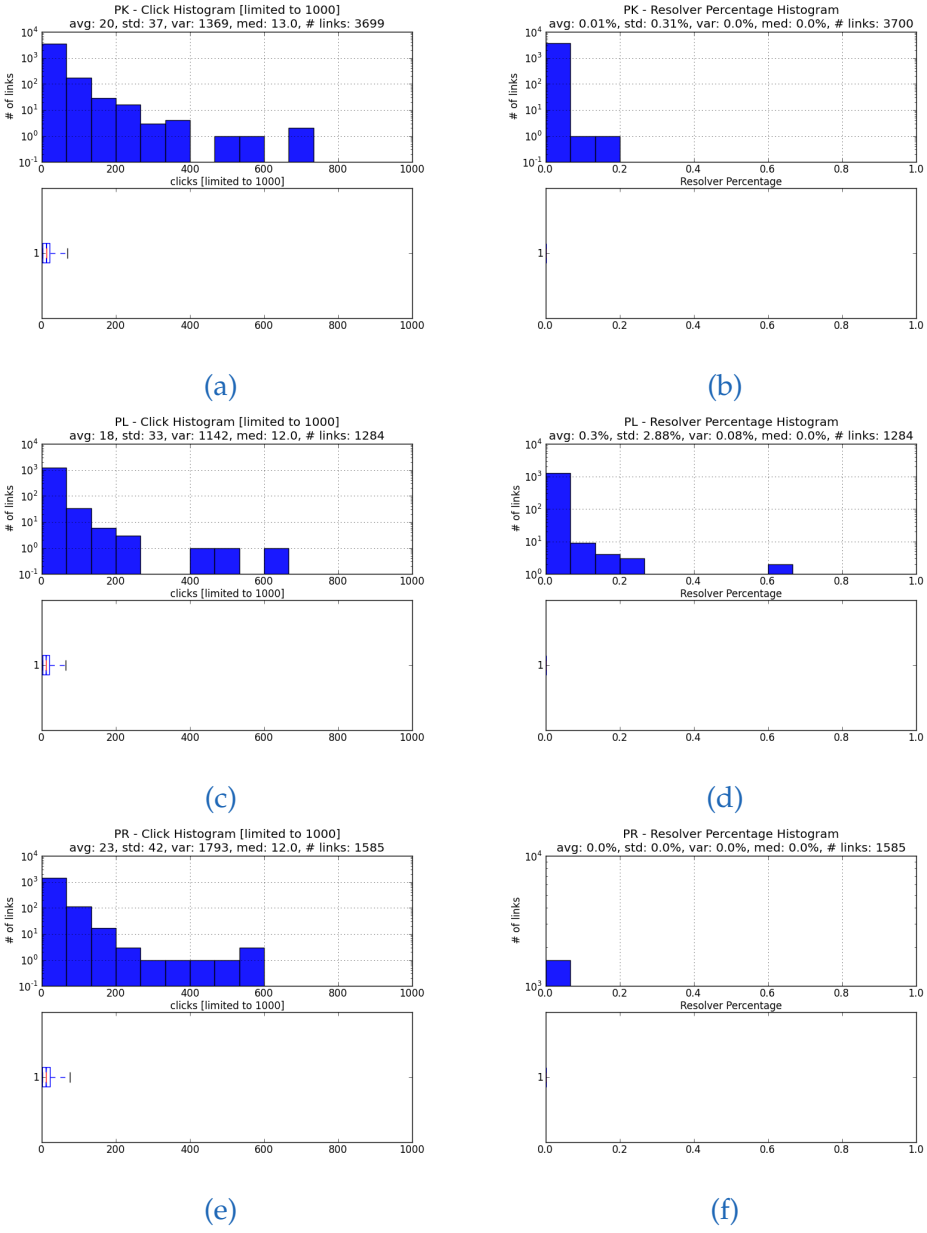



Figure B.13.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side.
(a): Click histogram for Pakistan. (b): Resolver Percentage histogram for Pakistan. (c): Click histogram for Poland. (d): Resolver Percentage histogram for Poland. (e): Click histogram for Puerto Rico. (f): Resolver Percentage histogram for Puerto Rico.

# Appendix B. Histogram Country Plots

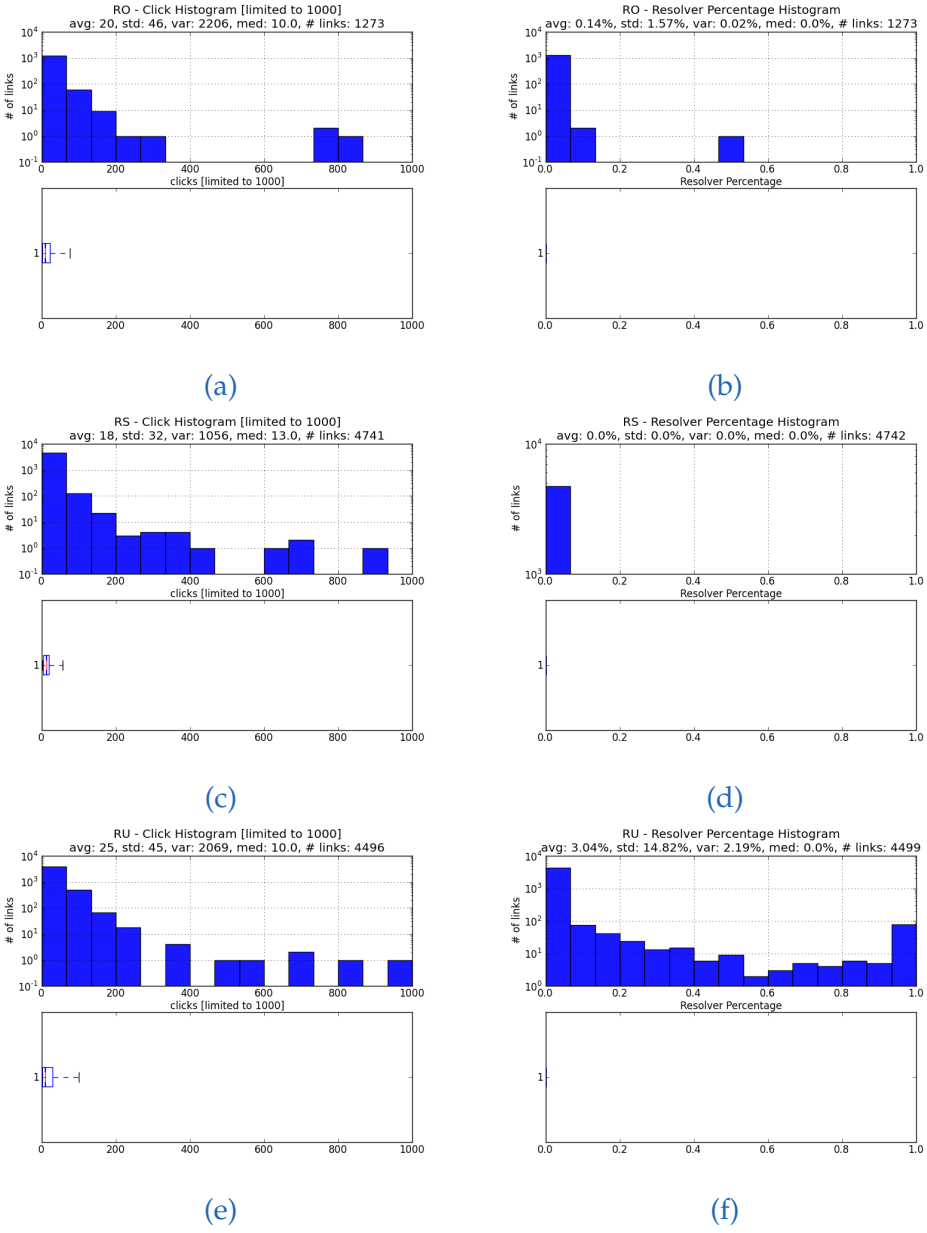
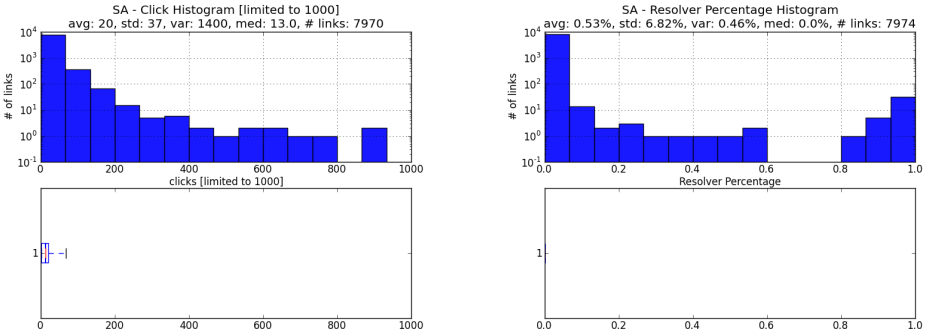



Figure B.14.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side.
(a): Click histogram for Romania. (b): Resolver Percentage histogram for Romania. (c): Click histogram for Serbia. (d): Resolver Percentage histogram for Serbia. (e): Click histogram for Russia. (f): Resolver Percentage histogram for Russia.

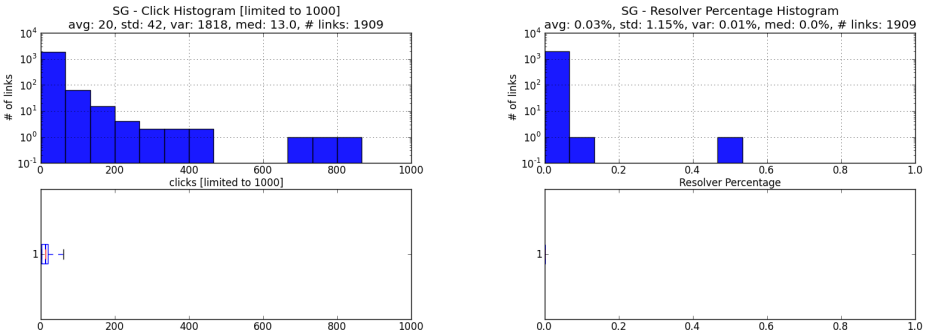
# Appendix B. Histogram Country Plots




(a)



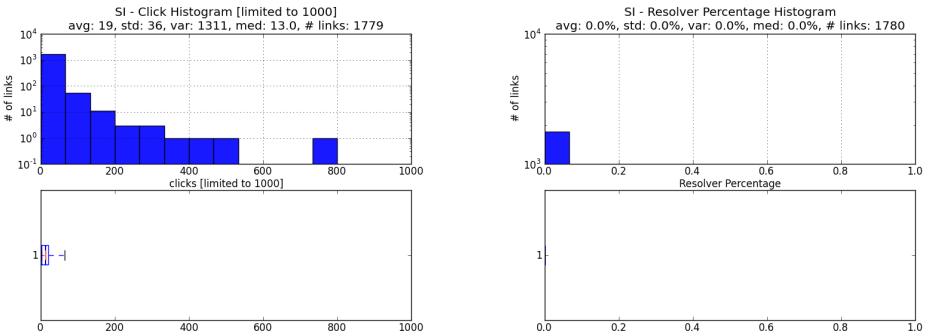

(b)




(c)




(d)



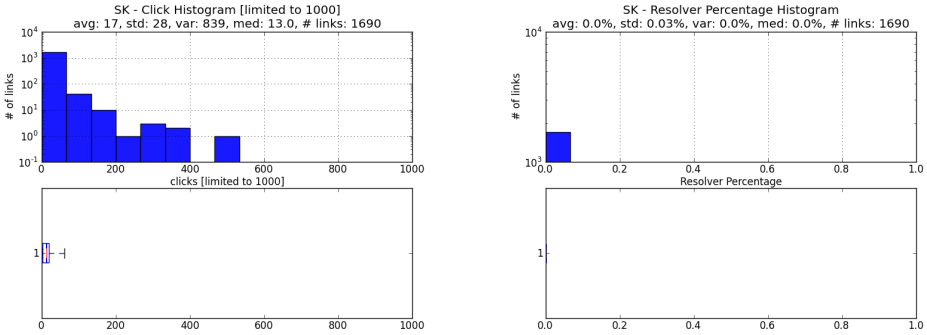

(e)




(f)

Figure B.15.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side.
(a): Click histogram for Saudi Arabia. (b): Resolver Percentage histogram for Saudi Arabia. (c): Click histogram for Singapore. (d): Resolver Percentage histogram for Singapore. (e): Click histogram for Slovenia. (f): Resolver Percentage histogram for Slovenia.

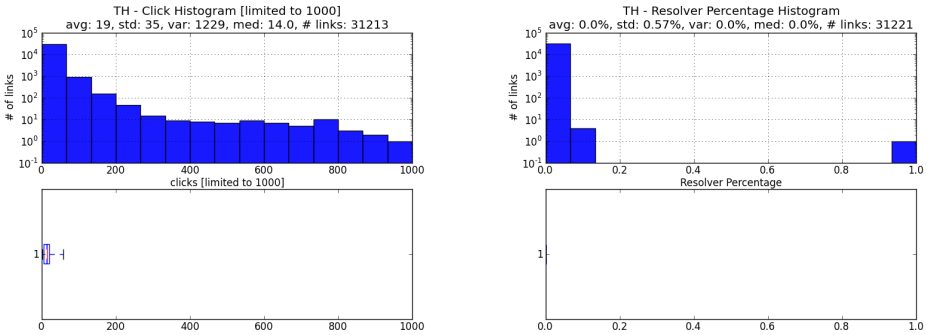
# Appendix B. Histogram Country Plots




(a)



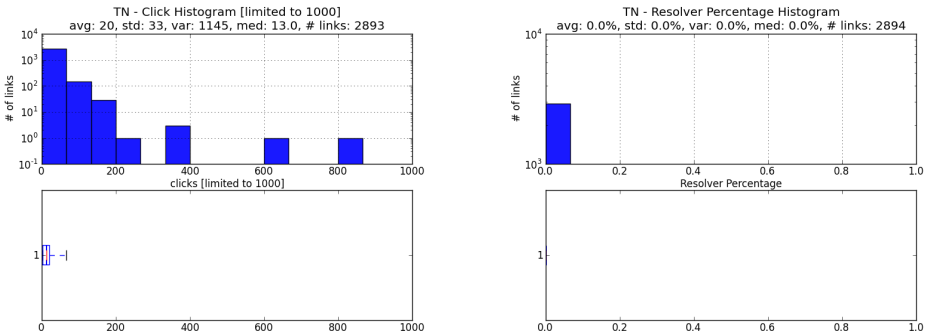

(b)




(c)




(d)



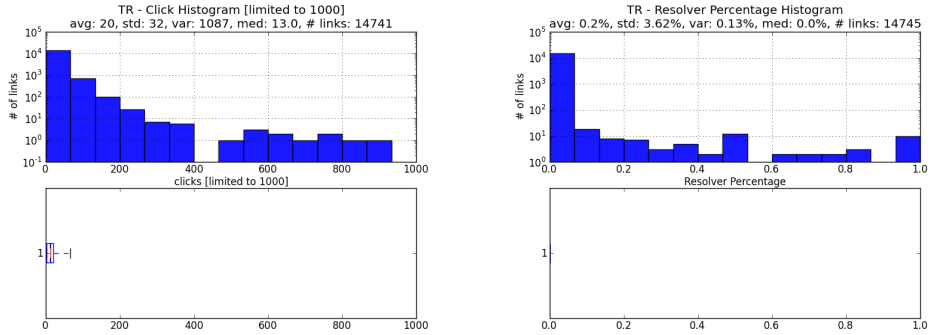

(e)




(f)

Figure B.16.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side.
(a): Click histogram for Slovakia. (b): Resolver Percentage histogram for Slovakia. (c): Click histogram for Thailand. (d): Resolver Percentage histogram for Thailand. (e): Click histogram for Tunisia. (f): Resolver Percentage histogram for Tunisia.

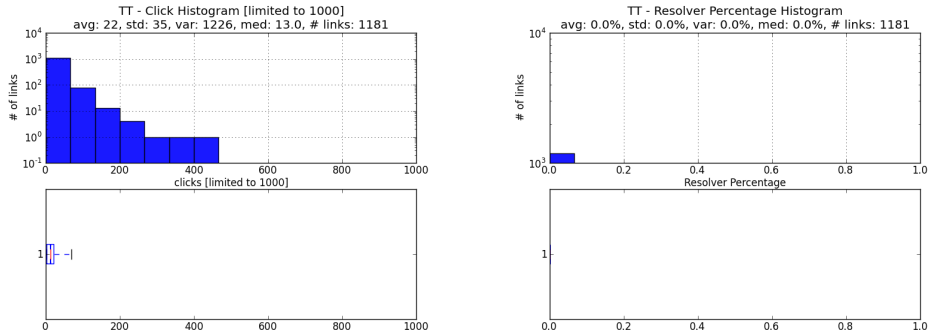
# Appendix B. Histogram Country Plots




(a)



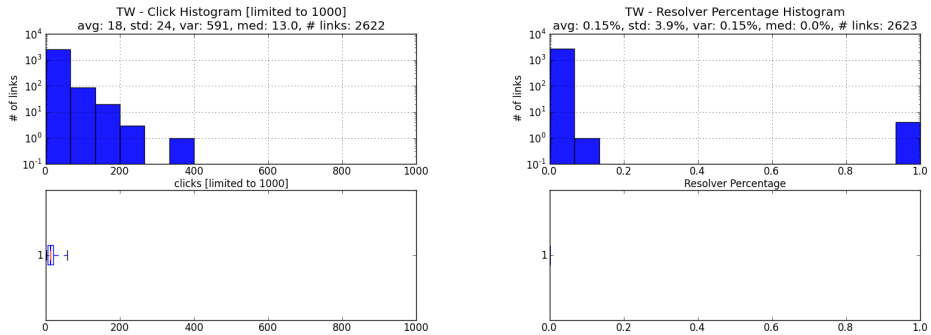

(b)




(c)




(d)




(e)




(f)

Figure B.17.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side.
(a): Click histogram for Turkey. (b): Resolver Percentage histogram for Turkey. (c): Click histogram for Trinidad and Tobago. (d): Resolver Percentage histogram for Trinidad and Tobago. (e): Click histogram for Taiwan. (f): Resolver Percentage histogram for Taiwan.

# Appendix B. Histogram Country Plots

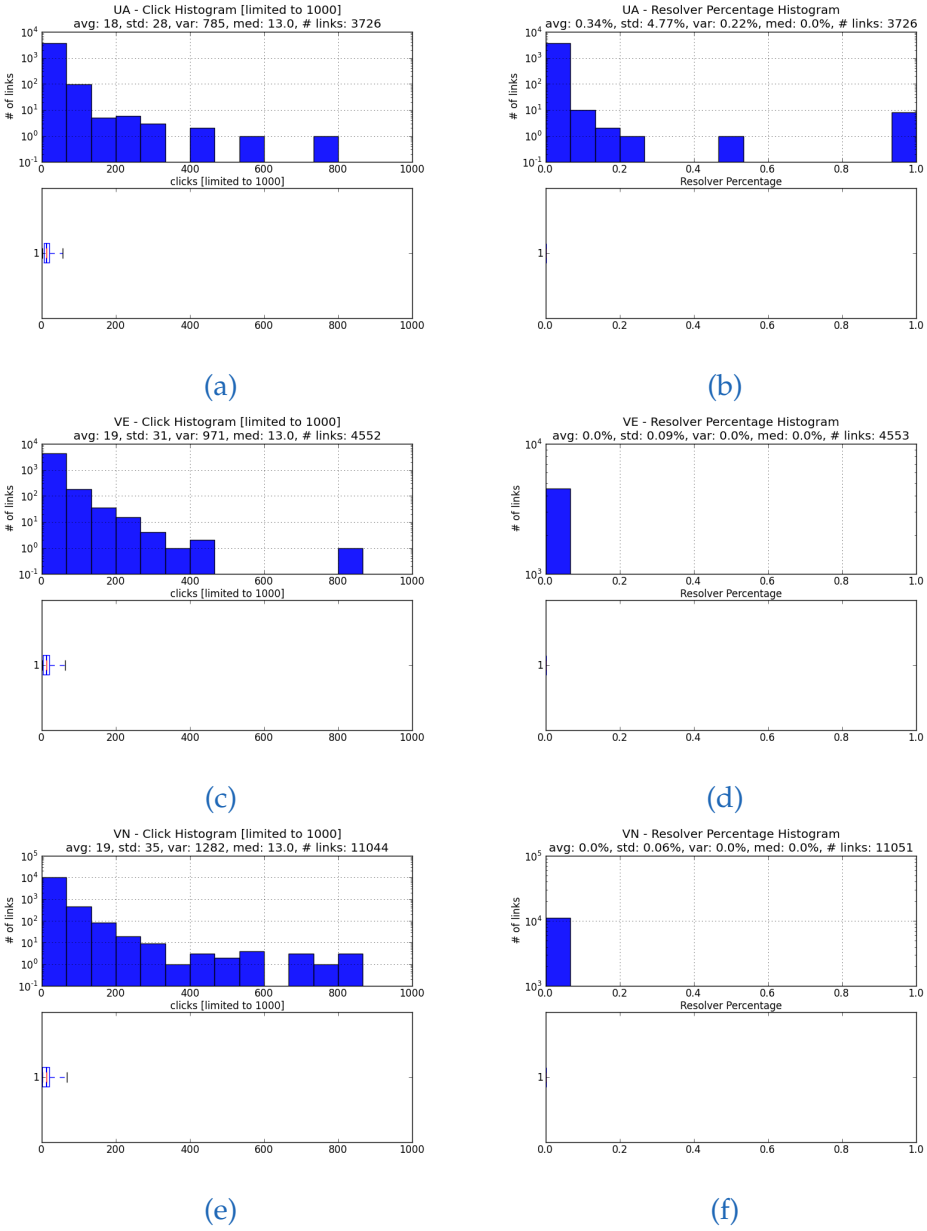



Figure B.18.: Click histogram for links created in different countries on the left hand side, Resolver Percentage histogram for locally created links on the right hand side.
(a): Click histogram for the Ukraine. (b): Resolver Percentage histogram for the Ukraine. (c): Click histogram for Venezuela. (d): Resolver Percentage histogram for Venezuela. (e): Click histogram for Viet Nam. (f): Resolver Percentage histogram for Viet Nam.

# Appendix C.

# Features

This section lists the feature importance that was gained from a recursive feature elimination with logistic regression. Table C.1 lists the feature importance of subset A, table C.2 the feature importance of subset B. Further table C.3 lists the 'ISO 3166-1 alpha-2' country codes used as label for the country binary features. Finally it shows the complete feature correlation matrix in figure C.1 on page 90.

Table C.1.: This table shows the rank of the features, applying RFE (recursive feature elimination) with Logistic Regression, on subset A.

| Rank | Feature | Rank | Feature | Rank | Feature |
|---|---|---|---|---|---|
| 1 | ctry_create_cnt | 2 | ctry_outdegree | 3 | click_time |
| 4 | ctry_indegree | 5 | domain_age | 6 | numip |
| 7 | minutes | 8 | localminutes | 9 | lon |
| 10 | ctry | 11 | lat | 12 | ctry_creator _percentage |
| 13 | fr | 14 | ctry_rc_ratio | 15 | br |
| 16 | ctry_resolver_percentage | 17 | de | 18 | eg |
| 19 | it | 20 | ae | 21 | il |
| 22 | kz | 23 | tr | 24 | in |
| 25 | sa | 26 | dz | 27 | mx |
| 28 | th | 29 | vn | 30 | za |
| 31 | tn | 32 | ph | 33 | my |
| 34 | ir | 35 | ma | 36 | jp |
| 37 | hk | 38 | ua | 39 | kr |
| 40 | ch | 41 | pr | 42 | at |
| 43 | al | 44 | sn | 45 | cl |

*Continued on next page*

Table C.1 – *Continued from previous page*

| Rank | Feature | Rank | Feature | Rank | Feature |
|---|---|---|---|---|---|
| **46** | pk | **47** | self_click | **48** | rs |
| **49** | ao | **50** | jm | **51** | ru |
| **52** | ke | **53** | si | **54** | gh |
| **55** | pt | **56** | us | **57** | hu |
| **58** | kw | **59** | cm | **60** | ge |
| **61** | pl | **62** | sk | **63** | tw |
| **64** | ga | **65** | mk | **66** | am |
| **67** | rw | **68** | gb | **69** | ro |
| **70** | bs | **71** | mz | **72** | hn |
| **73** | no | **74** | qa | **75** | es |
| **76** | bb | **77** | nl | **78** | cy |
| **79** | jo | **80** | md | **81** | ctry_irr |
| **82** | co | **83** | dk | **84** | ng |
| **85** | do | **86** | bg | **87** | ba |
| **88** | mn | **89** | vi | **90** | cz |
| **91** | bf | **92** | ag | **93** | id |
| **94** | ar | **95** | ug | **96** | fi |
| **97** | zw | **98** | py | **99** | lb |
| **100** | zm | **101** | fj | **102** | cn |
| **103** | iq | **104** | lk | **105** | sz |
| **106** | mw | **107** | bh | **108** | kg |
| **109** | ca | **110** | pe | **111** | np |
| **112** | lt | **113** | kh | **114** | tt |
| **115** | bo | **116** | mq | **117** | tj |
| **118** | ps | **119** | sy | **120** | uz |
| **121** | dj | **122** | be | **123** | lv |
| **124** | by | **125** | sd | **126** | lr |
| **127** | ci | **128** | bj | **129** | im |
| **130** | ve | **131** | ly | **132** | gt |
| **133** | gp | **134** | tc | **135** | lc |
| **136** | ec | **137** | ht | **138** | kn |
| **139** | au | **140** | bw | **141** | a2 |
| **142** | om | **143** | az | **144** | mv |
| **145** | sg | **146** | ie | **147** | pa |
| **148** | bd | **149** | ni | **150** | se |
| **151** | uy | **152** | cr | **153** | sv |
| **154** | gr | **155** | la | **156** | aw |
| **157** | as | **158** | ee | **159** | ky |
| **160** | hr | **161** | gf | **162** | gy |
| **163** | mu | | | | |

# Appendix C. Features

Table C.2.: This table shows the rank of the features, applying RFE (recursive feature elimination) with Logistic Regression, on subset B.

| Rank | Feature | Rank | Feature | Rank | Feature |
|---|---|---|---|---|---|
| 1 | domain_age | 2 | ctry_resolver_percentage | 3 | us |
| 4 | ctry_rc_ratio | 5 | ctry_indegree | 6 | fr |
| 7 | lon | 8 | it | 9 | ctry |
| 10 | lat | 11 | ctry_creator_percentage | 12 | click_time |
| 13 | numip | 14 | mx | 15 | th |
| 16 | ctry_create_cnt | 17 | ua | 18 | tr |
| 19 | my | 20 | co | 21 | kz |
| 22 | ar | 23 | br | 24 | ae |
| 25 | eg | 26 | ph | 27 | pe |
| 28 | kr | 29 | in | 30 | ir |
| 31 | sa | 32 | rs | 33 | ve |
| 34 | at | 35 | ctry_outdegree | 36 | ctry_irr |
| 37 | il | 38 | de | 39 | es |
| 40 | pk | 41 | ro | 42 | cl |
| 43 | dz | 44 | localminutes | 45 | jp |
| 46 | id | 47 | tw | 48 | hk |
| 49 | vn | 50 | ru | 51 | pl |
| 52 | bg | 53 | ec | 54 | jo |
| 55 | ma | 56 | do | 57 | sg |
| 58 | tn | 59 | hu | 60 | sk |
| 61 | ba | 62 | gr | 63 | jm |
| 64 | self_click | 65 | ch | 66 | qa |
| 67 | bo | 68 | si | 69 | gt |
| 70 | kw | 71 | nl | 72 | tt |
| 73 | ge | 74 | za | 75 | pr |
| 76 | al | 77 | gb | 78 | lb |
| 79 | se | 80 | hn | 81 | mk |
| 82 | sv | 83 | by | 84 | kg |
| 85 | pt | 86 | bh | 87 | ps |
| 88 | cy | 89 | be | 90 | pa |
| 91 | uy | 92 | uz | 93 | ht |
| 94 | sn | 95 | ly | 96 | minutes |
| 97 | ke | 98 | ee | 99 | ci |
| 100 | ni | 101 | lv | 102 | lt |
| 103 | cm | 104 | bd | 105 | az |
| 106 | ie | 107 | no | 108 | np |

*Continued on next page*

Table C.2 – *Continued from previous page*

| Rank | Feature | Rank | Feature | Rank | Feature |
|---|---|---|---|---|---|
| **109** | am | **110** | gh | **111** | sy |
| **112** | dk | **113** | cr | **114** | lk |
| **115** | rw | **116** | bs | **117** | sd |
| **118** | iq | **119** | md | **120** | bw |
| **121** | hr | **122** | ca | **123** | ao |
| **124** | fi | **125** | cz | **126** | zw |
| **127** | gy | **128** | ng | **129** | tj |
| **130** | lc | **131** | py | **132** | mz |
| **133** | ug | **134** | mu | **135** | bj |
| **136** | om | **137** | ga | **138** | nz |
| **139** | cw | **140** | gm | **141** | tz |
| **142** | vi | **143** | fj | **144** | sr |
| **145** | ag | **146** | is | **147** | mw |
| **148** | sz | **149** | ye | **150** | tc |
| **151** | sc | **152** | as | **153** | bf |
| **154** | vc | **155** | mp | **156** | me |
| **157** | mv | **158** | ml | **159** | bb |
| **160** | kh | **161** | cd | **162** | a2 |
| **163** | bt | **164** | bz | **165** | mg |
| **166** | pf | **167** | to | **168** | zm |
| **169** | dj | **170** | re | **171** | ne |
| **172** | pg | **173** | af | **174** | gp |
| **175** | cv | **176** | mo | **177** | au |
| **178** | bn | **179** | ls | **180** | sl |
| **181** | bm | **182** | lu | **183** | la |
| **184** | aw | **185** | dm | **186** | cg |
| **187** | kn | **188** | ai | **189** | ax |
| **190** | je | **191** | et | **192** | gd |
| **193** | gu | **194** | st | **195** | vu |
| **196** | gn | **197** | fm | **198** | eu |
| **199** | mn | **200** | tg | **201** | gq |
| **202** | lr | **203** | bi | **204** | er |
| **205** | ws | **206** | im | **207** | a1 |
| **208** | nc | **209** | cu | **210** | fo |
| **211** | ki | **212** | mc | **213** | gf |
| **214** | tm | **215** | vg | **216** | cn |
| **217** | na | **218** | mh | **219** | ms |
| **220** | cf | **221** | gi | **222** | mt |
| **223** | gl | **224** | km | **225** | mq |
| **226** | mr | **227** | ky | **228** | gg |

# Appendix C. Features

Table C.3.: This table shows the name of the countries represented by the two letter country code, used as label for the machine learning algorithms. Special codes such as 'A2' are used by GeoLite to identify IPs without geo-location.

| code | Name of country or region | code | Name of country or region | code | Name of country or region |
|---|---|---|---|---|---|
| **A1** | Anonymous Proxy | **A2** | Satellite Provider | **O1** | Other Country |
| **AD** | Andorra | **AE** | United Arab Emirates | **AF** | Afghanistan |
| **AG** | Antigua and Barbuda | **AI** | Anguilla | **AL** | Albania |
| **AM** | Armenia | **AO** | Angola | **AP** | Asia/Pacific Region |
| **AQ** | Antarctica | **AR** | Argentina | **AS** | American Samoa |
| **AT** | Austria | **AU** | Australia | **AW** | Aruba |
| **AX** | Aland Islands | **AZ** | Azerbaijan | **BA** | Bosnia and Herzegovina |
| **BB** | Barbados | **BD** | Bangladesh | **BE** | Belgium |
| **BF** | Burkina Faso | **BG** | Bulgaria | **BH** | Bahrain |
| **BI** | Burundi | **BJ** | Benin | **BL** | Saint Bartelemey |
| **BM** | Bermuda | **BN** | Brunei Darussalam | **BO** | Bolivia |
| **BQ** | Bonaire, Saint Eustatius and Saba | **BR** | Brazil | **BS** | Bahamas |
| **BT** | Bhutan | **BV** | Bouvet Island | **BW** | Botswana |
| **BY** | Belarus | **BZ** | Belize | **CA** | Canada |
| **CC** | Cocos (Keeling) Islands | **CD** | Congo, The Democratic Republic of the | **CF** | Central African Republic |
| **CG** | Congo | **CH** | Switzerland | **CI** | Cote d'Ivoire |
| **CK** | Cook Islands | **CL** | Chile | **CM** | Cameroon |
| **CN** | China | **CO** | Colombia | **CR** | Costa Rica |
| **CU** | Cuba | **CV** | Cape Verde | **CW** | Curacao |
| **CX** | Christmas Island | **CY** | Cyprus | **CZ** | Czech Republic |
| **DE** | Germany | **DJ** | Djibouti | **DK** | Denmark |
| **DM** | Dominica | **DO** | Dominican Republic | **DZ** | Algeria |
| **EC** | Ecuador | **EE** | Estonia | **EG** | Egypt |
| **EH** | Western Sahara | **ER** | Eritrea | **ES** | Spain |
| **ET** | Ethiopia | **EU** | Europe | **FI** | Finland |
| **FJ** | Fiji | **FK** | Falkland Islands (Malvinas) | **FM** | Micronesia, Federated States of |
| **FO** | Faroe Islands | **FR** | France | **GA** | Gabon |
| **GB** | United Kingdom | **GD** | Grenada | **GE** | Georgia |
| **GF** | French Guiana | **GG** | Guernsey | **GH** | Ghana |
| **GI** | Gibraltar | **GL** | Greenland | **GM** | Gambia |
| **GN** | Guinea | **GP** | Guadeloupe | **GQ** | Equatorial Guinea |
| **GR** | Greece | **GS** | South Georgia and the South Sandwich Islands | **GT** | Guatemala |
| **GU** | Guam | **GW** | Guinea-Bissau | **GY** | Guyana |
| **HK** | Hong Kong | **HM** | Heard Island and McDonald Islands | **HN** | Honduras |
| **HR** | Croatia | **HT** | Haiti | **HU** | Hungary |
| **ID** | Indonesia | **IE** | Ireland | **IL** | Israel |
| **IM** | Isle of Man | **IN** | India | **IO** | British Indian Ocean Territory |
| **IQ** | Iraq | **IR** | Iran, Islamic Republic of | **IS** | Iceland |
| **IT** | Italy | **JE** | Jersey | **JM** | Jamaica |
| **JO** | Jordan | **JP** | Japan | **KE** | Kenya |
| **KG** | Kyrgyzstan | **KH** | Cambodia | **KI** | Kiribati |
| **KM** | Comoros | **KN** | Saint Kitts and Nevis | **KP** | Korea, Democratic People's Republic of |
| **KR** | Korea, Republic of | **KW** | Kuwait | **KY** | Cayman Islands |
| **KZ** | Kazakhstan | **LA** | Lao People's Democratic Republic | **LB** | Lebanon |
| **LC** | Saint Lucia | **LI** | Liechtenstein | **LK** | Sri Lanka |
| **LR** | Liberia | **LS** | Lesotho | **LT** | Lithuania |
| **LU** | Luxembourg | **LV** | Latvia | **LY** | Libyan Arab Jamahiriya |
| **MA** | Morocco | **MC** | Monaco | **MD** | Moldova, Republic of |
| **ME** | Montenegro | **MF** | Saint Martin | **MG** | Madagascar |
| **MH** | Marshall Islands | **MK** | Macedonia | **ML** | Mali |
| **MM** | Myanmar | **MN** | Mongolia | **MO** | Macao |
| **MP** | Northern Mariana Islands | **MQ** | Martinique | **MR** | Mauritania |
| **MS** | Montserrat | **MT** | Malta | **MU** | Mauritius |
| **MV** | Maldives | **MW** | Malawi | **MX** | Mexico |
| **MY** | Malaysia | **MZ** | Mozambique | **NA** | Namibia |
| **NC** | New Caledonia | **NE** | Niger | **NF** | Norfolk Island |
| **NG** | Nigeria | **NI** | Nicaragua | **NL** | Netherlands |
| **NO** | Norway | **NP** | Nepal | **NR** | Nauru |
| **NU** | Niue | **NZ** | New Zealand | **OM** | Oman |

*Continued on next page*

Table C.3 – *Continued from previous page*

| code | Name of country or region | code | Name of country or region | code | Name of country or region |
|---|---|---|---|---|---|
| **PA** | Panama | **PE** | Peru | **PF** | French Polynesia |
| **PG** | Papua New Guinea | **PH** | Philippines | **PK** | Pakistan |
| **PL** | Poland | **PM** | Saint Pierre and Miquelon | **PN** | Pitcairn |
| **PR** | Puerto Rico | **PS** | Palestinian Territory | **PT** | Portugal |
| **PW** | Palau | **PY** | Paraguay | **QA** | Qatar |
| **RE** | Reunion | **RO** | Romania | **RS** | Serbia |
| **RU** | Russian Federation | **RW** | Rwanda | **SA** | Saudi Arabia |
| **SB** | Solomon Islands | **SC** | Seychelles | **SD** | Sudan |
| **SE** | Sweden | **SG** | Singapore | **SH** | Saint Helena |
| **SI** | Slovenia | **SJ** | Svalbard and Jan Mayen | **SK** | Slovakia |
| **SL** | Sierra Leone | **SM** | San Marino | **SN** | Senegal |
| **SO** | Somalia | **SR** | Suriname | **SS** | South Sudan |
| **ST** | Sao Tome and Principe | **SV** | El Salvador | **SX** | Sint Maarten |
| **SY** | Syrian Arab Republic | **SZ** | Swaziland | **TC** | Turks and Caicos Islands |
| **TD** | Chad | **TF** | French Southern Territories | **TG** | Togo |
| **TH** | Thailand | **TJ** | Tajikistan | **TK** | Tokelau |
| **TL** | Timor-Leste | **TM** | Turkmenistan | **TN** | Tunisia |
| **TO** | Tonga | **TR** | Turkey | **TT** | Trinidad and Tobago |
| **TV** | Tuvalu | **TW** | Taiwan | **TZ** | Tanzania, United Republic of |
| **UA** | Ukraine | **UG** | Uganda | **UM** | United States Minor Outlying Islands |
| **US** | United States | **UY** | Uruguay | **UZ** | Uzbekistan |
| **VA** | Holy See (Vatican City State) | **VC** | Saint Vincent and the Grenadines | **VE** | Venezuela |
| **VG** | Virgin Islands, British | **VI** | Virgin Islands, U.S. | **VN** | Vietnam |
| **VU** | Vanuatu | **WF** | Wallis and Futuna | **WS** | Samoa |
| **YE** | Yemen | **YT** | Mayotte | **ZA** | South Africa |
| **ZM** | Zambia | **ZW** | Zimbabwe | | |

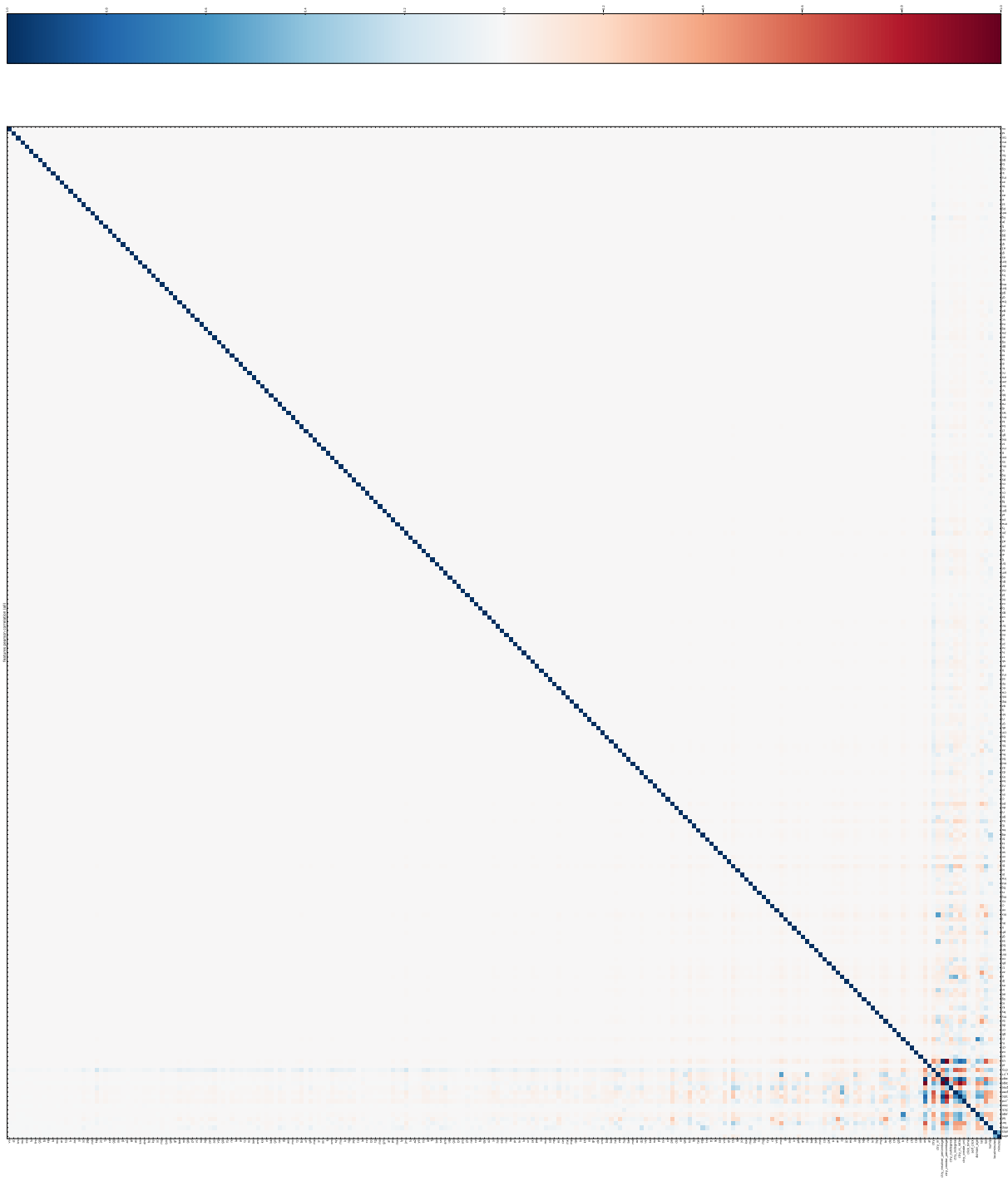


Figure C.1.: The full correlation matrix containing all features. A smaller version containing the top 16 features can be seen in figure 3.9

# Bibliography


[09] *Goo.gl Challenges Bit.ly as King of the Short*. http://bits.blogs.nytimes.com/2009/12/14/googl-challenges-bitly-as-king-of-the-short/. 2009 (cit. on p. 2).

[10a] *Facebook HipHop*. http://developers.facebook.com/blog/post/2010/02/02/hiphop-for-php--move-fast/. 2010 (cit. on p. 62).

[10b] *Google Safebrowsing*. http://code.google.com/apis/safebrowsing/. 2010 (cit. on p. 16).

[10c] *J. Wein. Joewein.de LLC - fighting spam and scams on the Internet.* http://www.joewein.net/. 2010 (cit. on p. 16).

[11a] *25 Billion Pieces of Spam*. http://blog.akismet.com/2011/04/08/25-billion-pieces-of-spam/. 2011 (cit. on p. 16).

[11b] *The Going Rate on the Black Market for Your E-Mail Address*. http://www.securitymanagement.com/news/00000025-going-rate-black-market-your-email-address-008950. 2011 (cit. on p. 19).

[11c] *WordPress Now Powers 22 Percent Of New Active Websites In The U.S.* http://techcrunch.com/2011/08/19/wordpress-now-powers-22-percent-of-new-active-websites-in-the-us/. 2011 (cit. on p. 16).

[12a] *Akismet, Wordpress Anti-Spam Plugin*. http://akismet.com/wordpress/. 2012 (cit. on p. 16).

[12b] *bit.ly*. Spam and Malware protection, http://blog.bitly.com/post/138381844/spam-and-malware-protection. 2012 (cit. on pp. 13, 21).

[12c] *qr.cx usage time analysis video*. http://qr.cx/8Ctq or http://youtu.be/06Mhn0L23Tk&hd=1. 2012 (cit. on p. 26).

[12d] *Safe.mn Safety FAQ*. http://safe.mn/faq/safety. 2012 (cit. on p. 21).

[12e] *Scumware.org*. http://www.scumware.org/report/91.218.39.245. 2012 (cit. on p. 20).

[12f] *URIBL*. URIBL.com, http://www.uribl.com/about.shtml – realtime URI blacklist. 2012 (cit. on pp. 16, 20).

[13a] *Google, Safe Browsing*. https://developers.google.com/safe-browsing/. 2013 (cit. on p. 20).

[13b] *hip-hop github website*. https://github.com/facebook/hiphop-php/wiki. 2013 (cit. on p. 62).

[13c] *PhishTank*. http://www.phishtank.com/index.php. 2013 (cit. on p. 20).

[13d] *Spamhaus, Domain Block List*. http://www.spamhaus.org/dbl/. 2013 (cit. on p. 20).

[13e] *Wepawet, About*. http://wepawet.iseclab.org/about.php. 2013 (cit. on p. 20).

[And+07] D. S. Anderson, C. Fleizach, S. Savage, and G. M. Voelker. *Spamscatter: Characterizing internet scam hosting infrastructure*. 2007 (cit. on p. 19).

[Ant+11] D. Antoniades, I. Polakis, G. Kontaxis, E. Athanasopoulos, S. Ioannidis, E. P. Markatos, and T. Karagiannis. "we.b: the web of short urls." In: *Proceedings of the 20th international conference on World wide web*. WWW '11. Hyderabad, India: ACM, 2011, pp. 715–724. ISBN: 978-1-4503-0632-4. DOI: http://doi.acm.org/10.1145/1963405.1963505. URL: http://doi.acm.org/10.1145/1963405.1963505 (cit. on p. 11).

[Bal+00] P. Baldi, S. Brunak, Y. Chauvin, C. A. F. Andersen, and H. Nielsen. "Assessing the accuracy of prediction algorithms for classification: an overview." In: *Bioinformatics* 16.5 (2000), pp. 412–424 (cit. on p. 45).

[Ben+08] F. Benevenuto, F. Duarte, T. Rodrigues, V. Almeida, J. M. Almeida, and K. W. Ross. "Understanding video interactions in YouTube." In: *Proceedings of the 16th ACM international conference on Multimedia*. ACM. 2008, pp. 761–764 (cit. on p. 13).

[Ben+09] F. Benevenuto, T. Rodrigues, V. Almeida, J. Almeida, and K. Ross. "Video interactions in online video social networks." In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, Vol. 5.4 (2009), p. 30 (cit. on p. 13).

[Ben+10a] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida. "Detecting spammers on twitter." In: *Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS)*. Vol. 6. 2010 (cit. on pp. 12, 13).

[Ben+10b] F. Benevenuto, T. Rodrigues, V. Almeida, J. Almeida, M. Gonçalves, and K. Ross. "Video pollution on the web." In: *First Monday*, Vol. 15.4 (2010), pp. 1–14 (cit. on p. 13).

[Ber89] T. Berners-Lee. *Information Management: A Proposal*. `http://www.w3.org/History/1989/proposal.html`. 1989 (cit. on p. 5).

[BFM05] T. Berners-Lee, R. Fielding, and L. Masinter. *RFC 3986, Uniform Resource Identifier (URI): Generic Syntax*. Ed. by Internet Engineering Task Force (IETF). Request For Comments (RFC). 2005. URL: `http://www.ietf.org/rfc/rfc3986.txt` (cit. on pp. 7, 8, 22, 24).

[Bre+99] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. "Web caching and Zipf-like distributions: Evidence and implications." In: *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 1. IEEE. 1999, pp. 126–134 (cit. on p. 11).

[Bus45] V. Bush. *As we may think*. 1945 (cit. on p. 5).

[Cal+08] P. H Calais, D. E. V. Pires, D. O. Guedes, W. Meira Jr, C. Hoepers, and K. Steding-Jessen. "A campaign-based characterization of spamming strategies." In: *Proceedings of the 5th Conference on e-mail and anti-spam (CEAS), Mountain View, CA*. 2008 (cit. on p. 13).

[Cas+07] C. Castillo, D. Donato, A. Gionis, V. Murdock, and F. Silvestri. "Know your neighbors: web spam detection using the web topology." In: *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. SIGIR '07. Amsterdam, The Netherlands: ACM, 2007, pp. 423–430. ISBN: 978-1-59593-597-7. DOI: 10.1145/1277741.1277814. URL: http://doi.acm.org/10.1145/1277741.1277814 (cit. on pp. 15, 16).

[CDN05] P. Chirita, J. Diederich, and W. Nejdl. "MailRank: using ranking for spam detection." In: *Proceedings of the 14th ACM international conference on Information and knowledge management*. CIKM '05. Bremen, Germany: ACM, 2005, pp. 373–380. ISBN: 1-59593-140-6. DOI: 10.1145/1099554.1099671. URL: http://doi.acm.org/10.1145/1099554.1099671 (cit. on p. 15).

[Chh+11] S. Chhabra, A. Aggarwal, F. Benevenuto, and P. Kumaraguru. "Phi.sh/$oCiaL: the phishing landscape through short URLs." In: *Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference*. CEAS '11. Perth, Australia: ACM, 2011, pp. 92–101. ISBN: 978-1-4503-0788-8. DOI: 10.1145/2030376.2030387. URL: http://doi.acm.org/10.1145/2030376.2030387 (cit. on p. 12).

[CL98] L. F. Cranor and B. A. LaMacchia. "Spam!" In: *Communications of the ACM*, Vol. 41.8 (1998), pp. 74–83 (cit. on p. 18).

[DG08] J. Dean and S. Ghemawat. "MapReduce: simplified data processing on large clusters." In: *Communications of the ACM*, Vol. 51.1 (2008), pp. 107–113 (cit. on p. 43).

[FFM04] M. J. Freedman, E. Freudenthal, and D. Mazieres. "Democratizing content publication with Coral." In: NSDI. 2004 (cit. on p. 11).

[FMN04] D. Fetterly, M. Manasse, and M. Najork. "Spam, damn spam, and statistics: Using statistical analysis to locate spam web pages." In: *Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS 2004*. ACM. 2004, pp. 1–6 (cit. on p. 13).

[Gri+10] C. Grier, K. Thomas, V. Paxson, and M. Zhang. "@spam: the underground on 140 characters or less." In: *Proceedings of the 17th ACM conference on Computer and communications security*. CCS '10. Chicago, Illinois, USA, 2010, pp. 27–37. ISBN: 978-1-4503-0245-6. DOI: http://doi.acm.org/10.1145/1866307.1866311. URL: http://doi.acm.org/10.1145/1866307.1866311 (cit. on pp. 13, 16).

[Ino+11] T. Inoue, F. Toriumi, Y. Shirai, and S. Minato. "Great east Japan earthquake viewed from a URL shortener." In: *Proceedings of the Special Workshop on Internet and Disasters*. SWID '11. Tokyo, Japan: ACM, 2011, 8:1–8:8. ISBN: 978-1-4503-1044-4. DOI: 10.1145/2079360.2079368. URL: http://doi.acm.org/10.1145/2079360.2079368 (cit. on pp. 10, 11).

[Joa98] T. Joachims. "Text categorization with support vector machines: Learning with many relevant features." In: *Machine learning: ECML-98* (1998), pp. 137–142 (cit. on p. 12).

[Kan+08] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G.M. Voelker, V. Paxson, and S. Savage. "Spamalytics: An empirical analysis of spam marketing conversion." In: *Proceedings of the 15th ACM conference on Computer and communications security*. 2008, pp. 3–14 (cit. on pp. 14, 15).

[Kle99] J. M. Kleinberg. "Authoritative sources in a hyperlinked environment." In: *Journal of the ACM (JACM)*, Vol. 46.5 (1999), pp. 604–632 (cit. on p. 17).

[KS12] F. Klien and M. Strohmaier. "Short links under attack: geographical analysis of spam in a URL shortener network." In: *Proceedings of the 23rd ACM conference on Hypertext and social media*. HT '12. Milwaukee, Wisconsin, USA: ACM, 2012, pp. 83–88. ISBN: 978-1-4503-1335-3. DOI: 10.1145/2309996.2310010. URL: http://doi.acm.org/10.1145/2309996.2310010 (cit. on pp. 4, 13).

[Mag+13] F. Maggi, A. Frossi, S. Zanero, G. Stringhini, B. Stone-Gross, C. Kruegel, and G. Vigna. "Two Years of Short URLs Internet Measurement: Security Threats and Countermeasures." In: *Proceeding of the 22nd international World Wide Web conference*.

WWW '13. Rio de Janeiro, Brazil: ACM, 2013. ISBN: 978-1-4503-2035-1/13/05 (cit. on pp. 2, 12, 13, 38).

[Mau96] H. Maurer. *HyperWave - The Next Generation Web Solution*. 1996 (cit. on p. 6).

[Pag+99] L. Page, S. Brin, R. Motwani, and T. Winograd. "The PageRank citation ranking: bringing order to the web." In: (1999) (cit. on p. 17).

[sur12] surbl. *SURBL - URI reputation data*. 2012. URL: `http://www.surbl.org` (cit. on pp. 20, 21).

[Tho07] A. Thomason. "Blog spam: A review." In: *Proceedings of Conference on Email and Anti-Spam (CEAS)*. 2007 (cit. on p. 13).

[WF94] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Structural analysis in the social sciences 8. Cambridge University Press, 1994. ISBN: 9780521387071 (cit. on p. 30).

[WGN06] I. H. Witten, M. Gori, and T. Numerico. *Web dragons: inside the myths of search engine technology*. Morgan Kaufmann, 2006 (cit. on pp. 6, 9, 17, 18).

[Wu+05] C. Wu, K. Cheng, Q. Zhu, and Y. Wu. "Using visual features for anti-spam filtering." In: *Image Processing. ICIP 2005. IEEE International Conference on*. Vol. 3. IEEE. 2005, pp. III–509 (cit. on p. 15).

[YL99] Y. Yang and X. Liu. "A re-examination of text categorization methods." In: *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 1999, pp. 42–49 (cit. on p. 45).