Evaluating Googlearchy: Assessing the Diversity of Political Websites

by

Brian Pitts

(Under the direction of Audrey Haynes)

Abstract

Websites are an increasingly important part of how people seek out political information. Many political scientists believe that the web provides diverse new sources of information. However, some research suggests that the structure of the web promotes a winners-take-all pattern wherein a few sites on any topic receive the bulk of the attention. By building software to analyze the links between webpages on a topic, I am able to demonstrate the latter pattern on four political issues. When analyzing the top sites on each issue, I find that traditional sources of information are outnumbered by web-only sources with formats that allow user participation.

Index words: search engine, hyperlink, internet, political communications

Evaluating Googlearchy: Assessing the Diversity of Political Websites

by

Brian Pitts

B.A., Emory University, 2007

A Thesis Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Arts

Athens, Georgia

2010

Evaluating Googlearchy: Assessing the Diversity of Political Websites

by

Brian Pitts

Acknowledgments

There are many people whose generosity in providing software for free modification and use reduced the amount of work I had to do to complete this project. In particular, I want to acknowledge Dr. Andrew McCallum for Rainbow, Connelly Barnes for the web_search python module, and Rev. Johnny Healey for sample web crawling code.

I am thankful to Dr. Audrey Haynes, Dr. Ryan Bakker, and Dr. Keith Dougherty for serving on my committee. I appreciate the time they took to offer me advice and guidance. I am particularly grateful to Dr. Haynes for pushing me to continue when my focus waned.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Political scientists have always been concerned with the characteristics of the sources of information within a democracy [2] [3]. Information plays a key role in a democracy for several reasons. The information citizens receive plays a role in the formation of their political preferences. Access to information determines a citizen's ability to monitor their government. Information about the political agenda is necessary for citizen participation. [9] Robert Dahl identifies information as a source of political advantage that can be more important than even wealth. The inequalities resulting from the uneven distribution of information, in his view, are a serious danger to democracy. The wider the spread of information, the less advantage elites have over the public. [13]

As sources of information expanded during our historical development, from the introduction of the pamphlet and newspaper, then radio and television, the critical focus on new sources has shifted from availability to quality to bias. Thus, the nature of these news sources, be they early pamphlets, partisan weeklies, the penny press, radio, television, and now the Internet, creates the particular concerns or hopes relating to the nature of information within our representative democracy. The concerns over homogeneity and bias found when our primary sources of news information, network news shows, dominated in the 1970s and 1980s, shifted to new hope for an egalitarian amalgamation of diverse news sources with the advent of the Internet in the 1990s. But that hope has found a counterpart in the critical view that the news to be found on the Internet, while generally more diverse, is dominated by a handful of popular sources. And this is of concern, as increasingly the populace is turning to the Internet for its political information. In prior times, as well, the primary role of arbiter of information was performed by journalists performing their task of scrutinizing the government's performance providing news on the events of the day as well as analysis of complex policy reforms. Today, that role is no longer held by journalists alone but by an array of news/information sources including interest groups, bloggers, wiki editors, and more.

The Internet, particularly the world wide web, is an increasingly important part of how people seek out political information and an important source of that information. According to results from a 2004 Pew/Michigan survey, 53% of US Internet users had gotten news about the Iraq war online, 35% of Internet users had gotten news about gay marriage online, and 26% of Internet users had gotten news about the debate over free trade online [17]. In a 2008 survey, 55% of the respondents went online to get information about the 2008 elections. In the same survey, 24% percent of Internet users reported getting news from issue-oriented websites. [34] Some of the most recent Pew surveys (April and December of 2009) suggest that 74% of American adults use the Internet. Of that 74%, 72% get their news online, while 60 % look online for news and information about politics or upcoming campaigns. The expectation is that these numbers will continue to grow. There is no doubt that the Internet is now a significant source of news and political information for a large number of Americans. [27]

Early theorists of the Internet championed it as an egalitarian medium; since the cost of producing a website is much lower than traditional publishing, and the potential reach of that website is much greater, the Internet was expected to expand the political voice and knowledge of the average citizen. As Howard Dean's campaign manager Joe Trippi effused, "The Internet is the most democratizing innovation we've ever seen, more so even than the printing press." [37]

The ability of traditional media to be the powerful agenda-setters of the past has been called into question with the arrival of this supposed diverse and interactive new medium [21] [36] [31]. Indeed, with such a diversity of information sources available to the public, the cyberoptimists also suggested the possibility of greater opportunities for citizen deliberation and participation [8] [11]. Schwartz (1996) and Rheingold (1993; 2002) argued that online communities and social networks could only augment the public sphere, and the expectation of greater civic engagement due to the free flow of ideas and diverse opinions would be enhanced by the Internet [7] [20].

Others have taken a more pessimistic view of the same phenomenon. Sunstein and Putnam, for example, fear that with public attention diffused across millions of websites political discourse will become more polarized [35] [28]. This is because such a high choice environment helps create clusters or niches of like-minded individuals. Thus, rather than being exposed to the generalized content of the traditional mass media, people find their sources and places on the Internet where they are "reinforcing each other's similarities" and "standing against people" of differing perspectives [33] [22] [25].

Another possibility, and one that is less examined, is that the Internet might not be so egalitarian after all. That it is not simply the individual's tendency to lessen cognitive dissonance that drives this polarization, but rather a structural explanation, or as some have suggested, an ecological one, drives the distribution of accessed websites. The dominant sources of news information on the web are simply not that diverse (i.e. they still fall within the general venue of mainstream political information), and more obscure and varied news sources are simply more difficult to find. [1]Thus, the primary question to be addressed here is whether or not the Internet is really an egalitarian structure when it comes to the most important type of information relative to a democratic nation —political news and information. Is there a diversity of political news and information on the web that is readily accessible to the public? If no, then why not?

Before addressing that question, however, the following chapter explores the literature that has informed this debate on the general question of the egalitarian nature of the web,

[1]An analogy might be certain sites are right there off the highway; the others require off-road maneuvering and if you are in a hurry, or lack the skills or motivation to go off-road, you take what you get, usually the first few of your search engines results.

and more precisely, whether political websites that are readily available to the public are diverse in nature.

Chapter 2

Literature Review

To understand why the Internet may be less diverse in terms of readily accessible information sources, it's necessary to reflect on the structure of the web. The element tying one webpage to another is the hyperlink. A hyperlink is a snippet of text or an image in a webpage that, when clicked by a computer user, causes their computer to load a new webpage. A hyperlink can be though of as having a source, the webpage that it is a part of, and a destination, the webpage it causes to be loaded when it is clicked. Across the web, a relatively small number of websites are the destination of the vast majority of hyperlinks. The distribution of hyperlinks to a particular site follows the highly skewed power law scale [6]. Figure 2.1 visually demonstrates a power law.

The distribution of the number of people visiting a particular site, commonly known as the "traffic" of a site, also follows a power law [5] [16]. To understand why site traffic should be related to hyperlink structure, it's necessary to think about the ways Internet users discover websites. If a user already knows about a website, they can visit it directly. If they don't, they can discover it via a hyperlink from a site they already know about or by using a search engine like Google. Both of these methods favor the discovery of highly linked-to sites.

Understanding how the following of hyperlinks favors highly linked-to sites is straightforward. When browsing the web, the more hyperlinks there are to a site the more likely a user is to come across and click one of them. The role of search engines is only slightly more complex. When using a search engine, most users only visit websites on the first page of results. The release of search data for over 600,000 AOL users showed that 90% of clicks

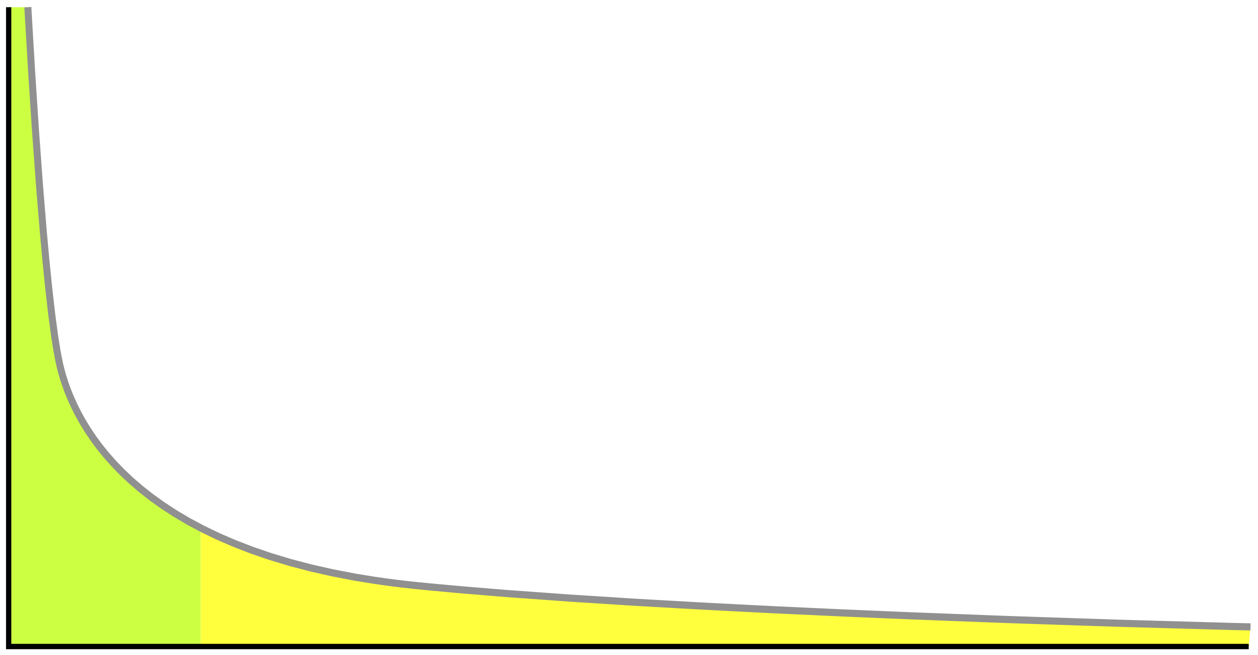


Figure 2.1: A power law distribution graphed on a linear scale. The number on the Y axis varies as a power of the X axis. The darker section represents 80% of the total items graphed. This image is by Hay Kranen and is in the public domain.

went to the results from the first page, 74% of clicks went to the first 5 results, and 42% of clicks went to the first result [23]. This is significant because search engines' rating algorithms give heavy weight to the number of hyperlinks a site receives. Although the exact algorithms vary from search engine to search engine and are often secret, Ding et al. show that search engine result ordering is barely distinguishable from simply ordering websites based on the number of hyperlinks to them [14]. Hindman found a .704 correlation between the amount of traffic a site received and the number of hyperlinks to it. [16]

Thus, hyperlinks to a website are a good measure of its visibility, and the visibility of a site is associated with the traffic it receives. [1]Hindman has dubbed this phenomena googlearchy, meaning "the rule of the most heavily linked". In a world of googlearchy, the rich sites get richer by accumulating more traffic and links at the expense of less visible sites. [16] Naming the phenomena after a search engine is apt, since it is users' reliance on search engines that accelerates this phenomenon. [1]

Does the existence of googlearchy on the web as whole also apply to political content? There is conflicting evidence in this regard. Looking purely at a set of 4,000 political weblogs [2], Drezner and Farrell found a highly skewed pattern of links. [15] Using techniques I will discuss in my methods section, Hindman examined communities of websites dealing with abortion, the death penalty, gun control, the presidency, the congress, and politics in general. In all of these cases, a power law fit the distribution of hyperlinks with an R^2 greater than .90.

However, other research on topically related websites shows a less skewed distribution. Pennock et al go as far to describe power law scaling within a category of websites as "the exception rather than the rule." Their study of links between company, university, and scientific webpages found lognormal distributions of links that differed sharply from the winners-take-all power law. When graphed on a log scale, Pennock's four topics differ in shape but all appear unimodal. Instead of winners and losers, there is a strong middle tier. [24] Chakrabarti et al examined links within hundreds of topics. Although they found that a power law tail was usually present, there were significant differences in the bodies of the distributions. [12] Thus, I believe the examination of more topics is necessary before accepting the idea that the visibility of websites on political issues follows a high skewed distribution such as a power law.

[1]Knowing that hyperlinks are a good proxy for traffic is extremely valuable because traffic is difficult for researchers to measure. Gathering data on the traffic of websites requires intercepting requests between a user's computer and the websites they visit. Internet service providers are the most reliable source of such information. A few large web analytics companies whose products are used by many websites are another source. Neither set of sources is regularly willing to release data at the scale or level of detail researchers would like. The analyses of traffic by Adamic and Huberman and Hindman both used data an Internet service provider gathered on their customers.

[2]A weblog, or blog, as defined by Drezner and Farrell is a "webpage with minimal to no external editing, providing on-line commentary, periodically updated and presented in reverse chronological order, with hyperlinks to other online sources." Another feature of blogs worth noting is that they frequently allow visitors to engage in a dialogue with the blog's author and other visitors through a comment section on each blog post.

Despite the Internet's importance, outside of examinations of blogs little research has been done on the sources of political information to which Internet users are most readily exposed. Hindman's research, for example, tells us that some sites are more visible than others but does not tell us anything about the characteristics of the most visible political websites. In contrast, I will examine characteristics of the most visible websites on an issue. For example, among these most visible, is there diversity in terms of focus and ideological leanings? Are they extensions of preexisting mainstream media or do they come from new sources? Answering these questions may have a real impact on how we understand the significance of the visibility of websites. Even if the distribution of links is highly skewed, the Internet might be broadening the range of political information the average citizen is exposed to if the most linked to sites convey information that is less visible in traditional media.

Thus I set out to accomplish two things. First, I will independently reimplement the techniques used by Hindman and use the tools I develop to test whether a highly skewed distribution of links, the key characteristic of googlearchy, exists in a set of political topics. Second, if I find such a distribution I will determine the sources, content, and format of the most visible sites. In the next section, I will discuss the data and methodology required to both test the googlearchy theory and further examine the nature and implications of this distribution in greater depth should I confirm its existence for these political topics.

Chapter 3

Methods

To gather the data necessary to answer these questions, I borrowed two techniques from computer science. The first is web crawling. Web crawling simply refers to starting at a webpage, observing all the hyperlinks it has to other webpages, visiting those other webpages, observing their hyperlinks, visiting them, and continuing this process an arbitrary number of times. A computer program that does this is called a web crawler. The second technique is automated text classification. It is not feasible for humans to classify the millions of webpages involved in a research project like this, but a variety of reliable statistical techniques have been developed that make it possible for computer software to conduct the analysis.

First, I decided upon a set of issues to examine. Hindman examined three issues: abortion, gun control, and the death penalty. I reexamine one of his issues, gun control, but also focus on gay marriage, global warming, and free trade. Including one of Hindman's issues allows me to verify his results. Including new issues allows me to see if his results were unique to the issues he studied. All four of the issues I examine are politically salient [1].

Second, I built a list of "seed" webpages for each issue. These webpages are important for two reasons. First, they define the starting points for the web crawler. Second, they are used to train the text classification system to recognize websites about an issue. In order to choose my seed pages, I developed a program that searched the Google, Yahoo, and MSN search engines for my four topics. For each topic, the program conducted the search on each search engine and collected the web addresses of the first 200 results from each engine. Next,

[1]In a LexisNexis search of Associated Press stories from 2009, gay marriage appeared in 2716 stories. global warming appeared in more than 3000 stories, gun control appeared in 1793 stories, and free trade appeared in 1133. LexisNexis will not return more than 3000 results.

it merged the results from the three search engines and saved the results in a file. Finally, it fetched the contents of each webpage that was returned in the search results and saved it in a file. Thus, once this program was done I had four files with web addresses to act as seeds in the crawler, one file for each topic. I also had four folders, each one with the contents of hundreds of webpages, to use for training the text classification system.

Third, I constructed a negative training set for the text classification system. This is a set of webpages that do not belong to any of my four categories. To choose a large set of webpages with known topics, I turned to the Open Directory Project. The Open Directory Project is a human-edited directory of the web. Webpages are assigned to categories, such as Arts:Television:Programs:Talk Shows or Society:Issues:Poverty:Hunger. I developed a program that took 600 randomly selected categories, retrieved 10,000 web addresses from them, and fetched the contents of each webpage. Once this program was done I had a file for each page.

Fourth, I trained the text classification software to distinguish between websites that are or are not related to an issue. The software I used was Rainbow, a part of Andrew McCallums' Bow toolkit. [2] Rainbow implements a naive Bayes classifier. [3]For each of my four issues, I ran Rainbow in indexing mode on the files with the contents of the seed webpages and the files with the contents of the negative training set. This produced four models that Rainbow could use to classify other webpages. Table 3.1 shows the ten words for each model that convey the most information about whether a document is relevant to the issue.

[2]I chose Rainbow because it has a number of features that made it easy to use. It has built in support for filtering out the markup language used to format webpages. It includes a “stoplist” of common words (such as “a” and “the”) that it will ignore. These features improve the accuracy of the text classification models constructed by Rainbow, and their inclusion in Rainbow allowed me to avoid preprocessing the data.

[3]Bayesian classifiers are often used for text classification problems due to their simplicity and effectiveness. For example, Bayesian classifiers are commonly used to filter spam email messages. Essentially they answer the question “Given these words observed in a document, what is the probability that it belongs to each category?” Before a classifier can be used it has to be trained with sample documents known to fit in each category of interest. The classifier stores the frequency of words found in each category and uses this to determine the probabilities for documents not in the training set.

Table 3.1: Top Words in Models

Information	Word
0.15	gay
0.15	marriage
0.12	sex
0.11	couples
0.09	unions
0.09	marry
0.09	marriages
0.09	gays
0.08	civil
0.07	lesbian

(a) Gay Marriage

Information	Word
0.16	warming
0.11	global
0.10	climate
0.10	greenhouse
0.10	carbon
0.09	dioxide
0.08	gases
0.08	emissions
0.08	scientists
0.07	earth

(b) Global Warming

Information	Word
0.16	gun
0.14	guns
0.12	firearms
0.11	control
0.10	amendment
0.09	laws
0.09	weapons
0.08	ban
0.08	crime
0.08	rifle

(c) Gun Control

Information	Word
0.12	trade
0.07	agreements
0.07	tariffs
0.06	agreement
0.06	nafta
0.06	free
0.06	economic
0.06	economy
0.06	countries
0.06	foreign

(d) Free Trade

Finally, I developed a web crawler and ran it for each issue. The web crawler started at each of the seed webpages, visited the webpages linked to by the seed webpages, and then visited all the webpages linked to by those pages. [4]The crawler ran the text of each webpage it visited through the text classification software to determine if it was related to the issue. If the webpage was unrelated, no hyperlinks from it were followed. For each webpage visited the crawler recorded the text classification score and the webpages to which it linked.

[4]Limiting the websites visited to those that are within 2 hyperlinks of the seeds was necessary due to computational constraints. Increasing the depth of a single crawl to 3 would have meant

visiting tens of millions of pages and processing hundreds of gigabytes of data. This would require months of computer time and significantly increase the costs of the research.

Chapter 4

Results

4.1 Crawl Overview

After performing the four crawls, I filtered out all data from websites that did not have a single relevant webpage. Table 4.1 demonstrates the scope of the four crawls after that filtering. The number of pages crawled ranged from 600,000 to 1,500,000. For each crawl, around one-sixth of the pages linked to were not found. This could be due to an error in the link (such as a misspelling), an error on the website (such as being overloaded), or a change in the site's content (perhaps the page was deleted). Also, a small number of pages were downloaded but had errors that prevented the crawler or classifier from processing them successfully. In both of these cases the crawler retried two times in case the errors were intermittent. The number of pages crawled is inflated compared to the amount of unique content crawled. While each page is defined by a unique web address, sometimes more than one web address can point to the same content. [1]

Table 4.1: Number of Pages Crawled

Topic	Crawled	Missing	Error
Gay Marriage	1,499,429	345,305	2,238
Global Warming	605,222	97,568	2,596
Gun Control	1,175,237	243,916	3,907
Free Trade	665,959	107,496	803

[1] For example, a blog post like `ablog.com/post.php` might also be linked to as `ablog.com/post.php?utm_source=feedburner`. This suggests that ignoring the parameters after the question mark might be a way to prevent duplicate content. However, this is not the case since many websites use such parameters to dynamically choose what content to display. For example, `ablog.com/post.php?id=5` and `ablog.com/post.php?id=4` could be different content.

Table 4.2 shows the aggregate results of the classification of the pages for each topic. For gay marriage, three-fourths of the pages successfully crawled were classified as relevant. For the other topics, two-thirds of the pages successfully crawled were classified as relevant.

Table 4.2: Page Classifications

Topic	Classified	Relevant	Irrelevant
Gay Marriage	1,151,886	871,448	280,438
Global Warming	505,058	336,000	169,058
Gun Control	927,414	617,876	309,538
Free Trade	557,660	380,428	177,232

Although each crawl encompassed hundreds of thousands of webpages, these webpages were a part of a much smaller number of websites. [2]The number of sites for each topic is shown in Table 4.3. This table also shows the number of links found between webpages. Throughout this analysis, I will make a distinction between all links and external links. The descriptor “all links” includes any links from one webpage to another. The descriptor “external links” only includes links from a webpage to a webpage on another website. This is distinct from “internal links”, which are links from a webpage to a webpage on the same website. For example, a link from `foo.com/news` to `bar.com/views` would be counted both in all links and external links. However, a link from `foo.com/news` to `foo.com/archive` would be counted in all links but not in external links. The reason for analyzing these two categories of links separately is that the inclusion of internal links might not accurately reflect the visibility of a website. For example, a website could be linked to only once by another website, but it could link to its own pages thousands of times. Table 4.3 shows that the number of external links is much smaller than the number of internal links.

[2]The count of websites is inflated somewhat, since every unique domain is treated as a site. For instance, blog.foo.com and forum.foo.com would be treated as two different websites, even though they would probably be under the control of the same individual or group. Collapsing the concept of a site down to the base domain would solve this problem, but it would introduce an undercount problem. Many blog hosting services use different subdomains for different customers, such as jane.livejournal.com or joe.livejournal.com. It would be undesirable to treat these as the same website.

Table 4.3: Number of Sites and Links

Topic	Sites	All Links	External Links
Gay Marriage	27,612	357,359,575	69,948,869
Global Warming	10,692	95,840,731	16,568,170
Gun Control	24,356	197,240,465	45,801,846
Free Trade	9,038	83,118,541	11,033,386

4.2 A Look at Links

Tables 4.4 and 4.5 show the number of links to the top 1, and 50 most linked-to sites for each category of links. In all cases, they reveal a highly-skewed distribution of links. Out of thousands of sites, the top 10 average almost 50% of all links and 25% of external links.

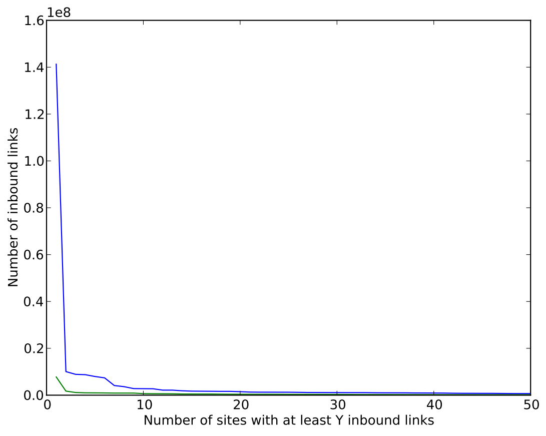
Table 4.4: Percentage of All Links to Top Sites

Topic	Links to Top 1	Links to Top 10	Links to Top 50
Gay Marriage	40%	55%	69%
Global Warming	20%	45%	71%
Gun Control	10%	40%	58%
Free Trade	21%	50%	73%

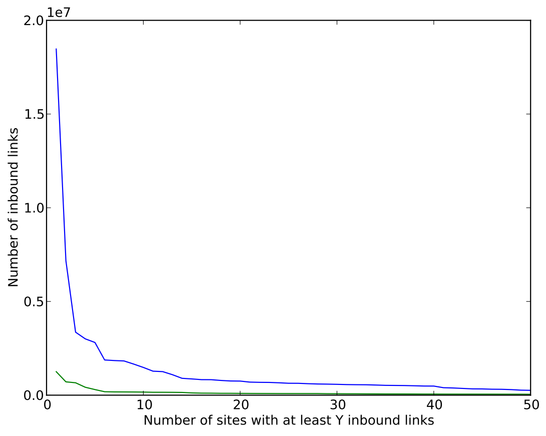
Table 4.5: Percentage of External Links to Top Sites

Topic	Links to Top 1	Links to Top 10	Links to Top 50
Gay Marriage	11%	24%	43%
Global Warming	8%	25%	45%
Gun Control	7%	22%	39%
Free Trade	10%	27%	49%

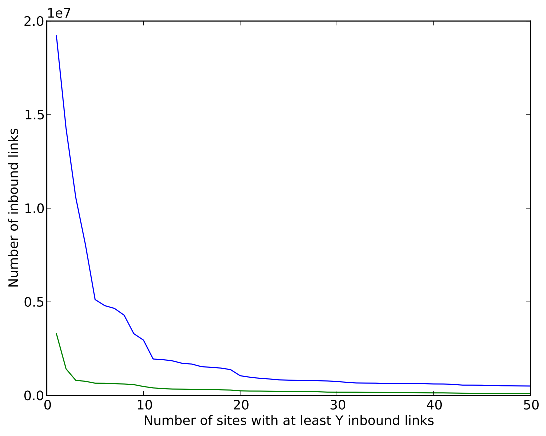
Is this highly-skewed distribution a power law, as previous research suggests? I perform both visual and statistical tests. First, Figure 4.1 show the links to the top 50 sites on a linear scale. Because of the long tail, graphing all sites would hide the gradual change in the top sites. The shape is similar to the sample graph in Figure 2.1.



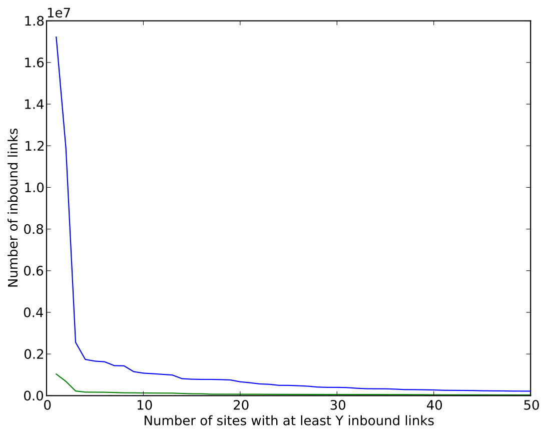
(a) Gay Marriage



(b) Global Warming



(c) Gun Control



(d) Free Trade

Figure 4.1: Linear Graphs of Top 50 Sites

The other tests rely on a useful property of power law distributions - when graphed with both axes on a log scale the data forms a straight line. Table 4.6 shows the result of an ordinary least squares regression with the log of the number of links to a website as the dependent variable and the log of the number of websites with as many hyperlinks as that website as the independent variable. For all topics, using either category of links, the model's fit is extremely good: the lowest R^2 is .95. However, this is not sufficient to prove that there is a linear relationship on a log-loq scale. Q-Q plots of the residuals against the normal distribution in Figure 4.2 shows a strongly nonlinear pattern.

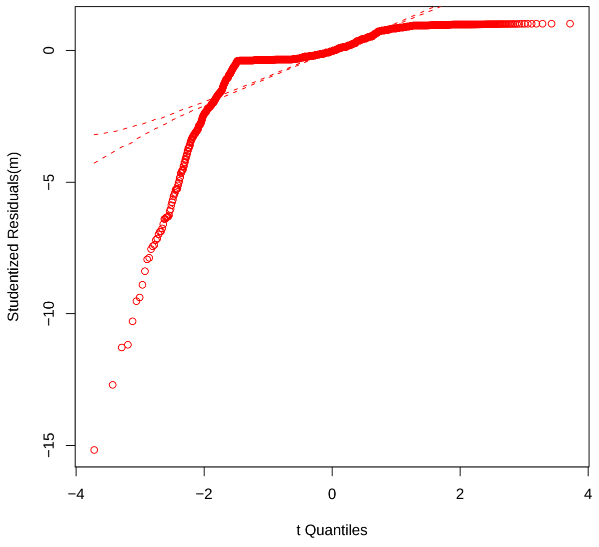
Table 4.6: Regression Fit

	All Links	External Links
Topic	R^2	R^2
Gay Marriage	.976	.975
Global Warming	.978	.950
Gun Control	.975	.960
Free Trade	.978	.958

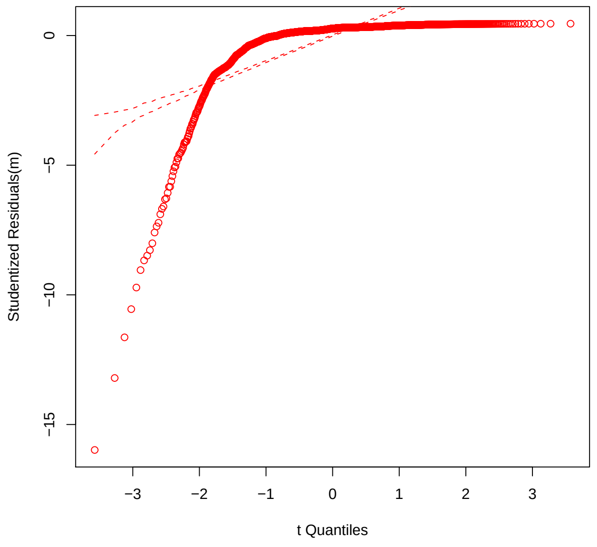
Figure 4.3 shows graphs of the data on a log-log scale for each topic. All links are in blue, and external links only are in green. The graphs again show the highly skewed distribution of links, but they also do not support a power law distribution across the entire spectrum of sites. This is particularly clear at the tail of least linked-to sites.

4.3 A Look at the Top Sites

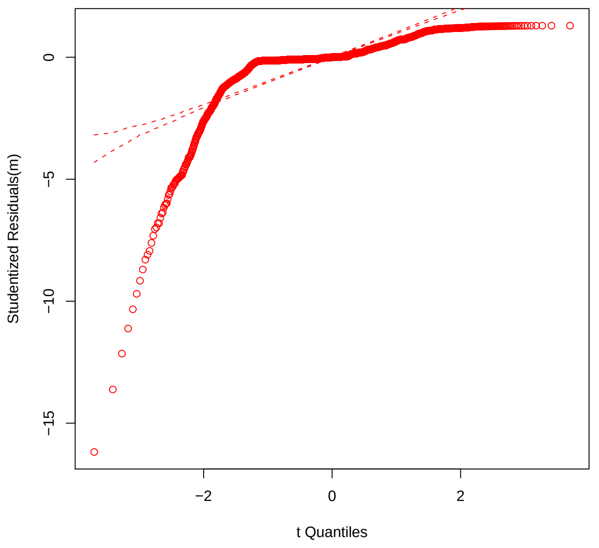
If the distribution of links makes some sites much more visible than others, an important question becomes what kind of sites are most visible. I examined the top 10 sites by all links in each category. [3]Tables 4.7 through 4.10 show the top 10 sites for each topic. The first 4 columns show data determined by the crawling software-the site names, the the number of links to them, the number of pages from them that were crawled, and the fraction of their pages that were relevant to the topic. The 5th column shows the focus of the site, determined



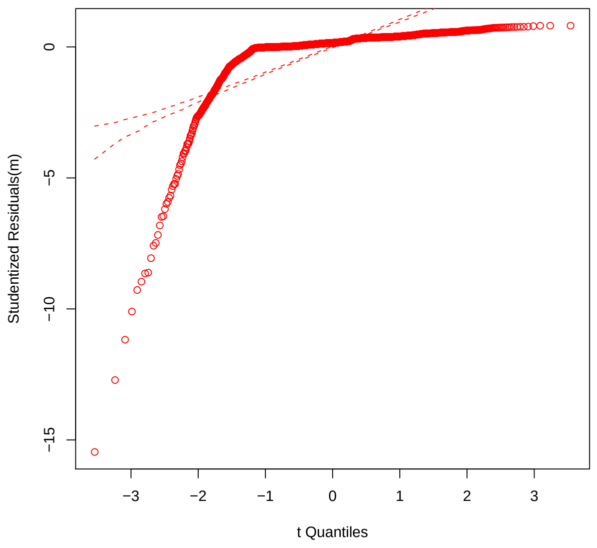
(a) Gay Marriage



(b) Global Warming

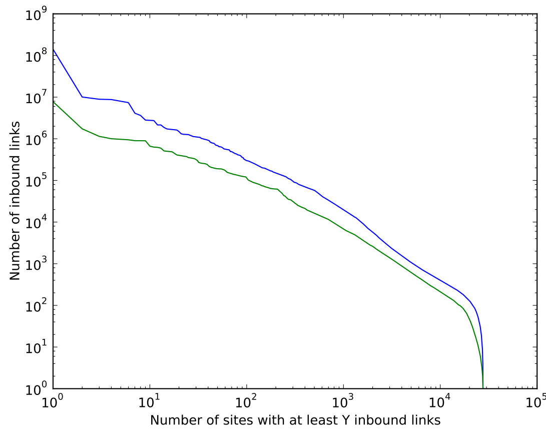


(c) Gun Control

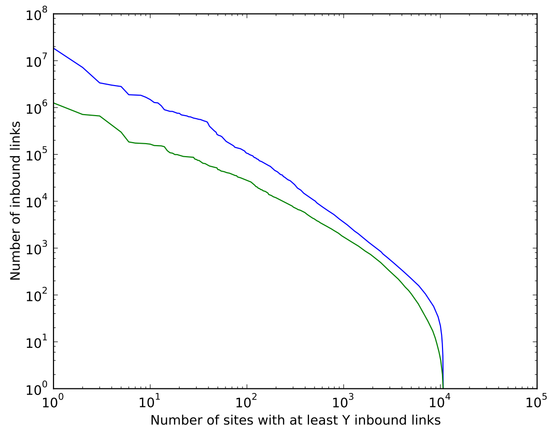


(d) Free Trade

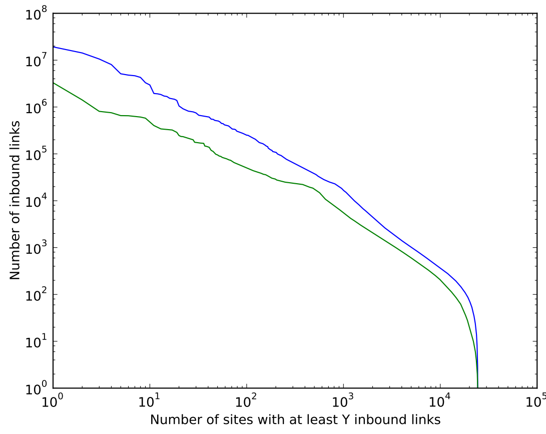
Figure 4.2: Q-Q Plots



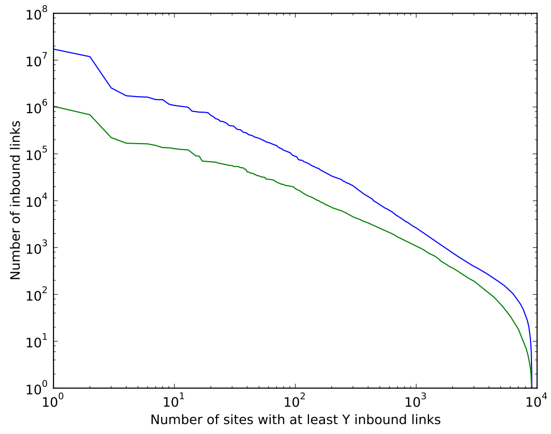
(a) Gay Marriage



(b) Global Warming



(c) Gun Control



(d) Free Trade

Figure 4.3: Log-log Graphs of All Sites

by viewing the content on the front page and by visiting the site's "About" page if it had one. The 6th column shows whether the site was either a blog or a wiki.

Surprisingly, there are a very small number of issue-specific sites in the top 10 for each category. For gay marriage, there is not a single site in the top 10 devoted to that issue. Expanding the scope to all gay rights issues yields only one site, pinknews.co.uk, that is exclusively devoted to covering them. On global warming, joannenova.com.au is the only site primarily focused on the topic. For gun control two sites, saysuncle.com and guncite.com, focus on it. Similar to gay marriage, free trade does not have any sites devoted to it in the top 10. The closest are knowmore.org, which compiles critical reports on corporations, and globalenvision.org, which reports on the effects of globalization on poverty. Instead of topic-specific sites, more general news sites are the dominant category. 10 of the 40 sites focus on political news, another 8 focus on other kinds of news.

The sources of the sites are interesting in who is absent. Although news sites are almost half of the total, online.wsj.com is the only one that is an online outgrowth of a print publication. Interest groups focused on the issues are completely absent from the lists. Only a single think tank, the Mises Institute, is represented.

Although blogs are a relatively new phenomenon on the web, they make a strong showing among the top sites. 20 of the 40 sites are completely in the blog format or feature a blog section prominently on their front page. An even more recent phenomenon is the wiki, a format where the entire site's structure and constant is collaboratively generated by its users. 3 of the 40 sites are wikis. Although not strictly a wiki, answers.com operates similarly. Its users can post questions, which are then viewable and answerable by other users. It appears as the top site in 3 of the 4 categories.

[3] An analysis using only external links was also conducted. Even fewer topical sites were included, and the top 10 sites had a lower relevance.

Table 4.7: Top Sites for Gay Marriage

Site	All Links	Pages	Relevance	Focus	Blog/Wiki
www.americablog.com	141287627	75469	0.81	Political News	Blog
graphjam.com	10064920	9691	0.99	Humor	Blog
www.fivethirtyeight.com	8894861	12670	0.95	Political News	Blog
www.amazon.com	8745562	69776	0.39	Shopping	No
www.blogger.com	7974559	90863	0.54	Various	Blog
www.huffingtonpost.com	7386997	23537	0.84	Political News	Blog
current.com	4115656	9776	0.91	General News	No
www.pinknews.co.uk	3637509	5873	0.98	Gay News	No
slate.com	2811210	14633	0.98	General News	No
donklephant.com	2770320	7701	0.76	Political News	Blog

Table 4.8: Top Sites for Global Warming

Site	All Links	Pages	Relevance	Focus	Blog/Wiki
www.answers.com	51594	18470009	0.68	Q&A	No
www.dailytech.com	8189	7175008	0.86	Technology News	Blog
joannenova.com.au	7953	3360679	0.86	Global Warming	Blog
www.conservapedia.com	10123	3005508	0.77	Conservative Issues	Wiki
www.huffingtonpost.com	11645	2811964	0.63	Political News	Blog
www.wikiality.com	9619	1878093	0.94	Humor	Wiki
www.sciencedaily.com	5598	1847411	0.91	Science News	No
www.sourcewatch.org	11341	1826653	0.87	Liberal Issues	Wiki
www.bookrags.com	9934	1659569	0.56	General Knowledge	No
www.britannica.com	5815	1479964	0.68	General Knowledge	No

Table 4.9: Top Sites for Gun Control

Site	All Links	Pages	Relevance	Focus	Blog/Wiki
www.answers.com	38082	19210496	0.62	Q&A	No
www.guncite.com	15871	14249227	1.0	Gun Rights	Blog
atlasshrugs2000.typepad.com	30384	10573786	0.76	Conservative Issues	Blog
www.swamppolitics.com	45610	8053263	0.85	Political News	Blog
www.amazon.com	50331	5122581	0.33	Shopping	No
rsmccain.blogspot.com	20204	4796587	0.93	Political News	Blog
www.americablog.com	2318	4650048	0.85	Political News	Blog
www.huffingtonpost.com	18397	4287929	0.61	Political News	Blog
www.blogger.com	25863	3300900	0.02	Various	Blog
www.saysuncle.com	22001	2956435	0.71	Gun Rights	Blog

Table 4.10: Top Sites for Free Trade

Site	All Links	Pages	Relevance	Focus	Blog/Wiki
www.answers.com	42552	17220402	0.72	Q&A	No
www.amazon.com	80756	11880226	0.49	Shopping	No
online.wsj.com	7341	2558830	0.93	General News	No
www.huffingtonpost.com	7379	1739235	0.67	Political News	Blog
www.knowmore.org	6311	1657288	1.0	Corporations	No
www.newsvine.com	11626	1626273	0.66	General News	No
www.britannica.com	5900	1440839	0.62	General Knowledge	No
www.globalenvision.org	6707	1432712	0.81	Global Poverty	Blog
www.businessinsider.com	3239	1152459	0.86	Business News	No
mises.org	5957	1081100	0.45	Economics	Blog

Chapter 5

Conclusion

5.1 Implications

The results of this research are supportive of the findings of Hindman and other researchers who have found highly skewed distributions at work in topical communities across the world wide web. My results were consistent across each topic and category of links: a tiny number of the sites receive the majority of the links, and much of the decline in links appears exponential. This winners-take-all pattern seems to be an inescapable effect of how pages are connected on the web. No matter how numerous and diverse the set of websites about an issue is, information seekers are funneled into a small number of websites.

For those who were hoping that the web would be a more egalitarian medium, the news is not completely bleak. Although it is governed by googlearchy, the rulers are fresh faces. Rather than online incarnations of print or television media, or political actors like interest groups or parties, bloggers and web-only news companies are the dominant destinations on the issues I examined. Combined, blogs and wikis are more than half of the top sites in each category. Since many more users can contribute and comment on blogs and wikis compared to traditional websites, there may be more viewpoints visible than the concentration of attention on such a small number of sties would suggest. However, it's important not to overstate this claim. That these kinds of sources are highly visible does not mean that the messages being conveyed are any different than those that would have been found by information seekers not using the web.

5.2 Future Research

With the data the software I have developed can gather, there are many other research questions that could be investigated.

5.2.1 Are the viewpoints of the most visible websites representative of the entire set of websites on an issue?

To answer this, the text of relevant webpages discovered by the crawler could be run through software to determine the ideology, such as Proksch and Slapin's Wordfish software. Wordfish estimates political position on a single-dimension through a statistical model of word counts. It would be interesting to compare the range of the Wordfish scores across the most visible websites to the range across the entire set. Similar, one could divide the Wordfish scale into bins and compare the percentage of sites in each bin for the most visible websites to the percentage in each bin across all sites. Also, it would be interesting to determine if there is a relationship between the Wordfish score and number of hyperlinks to a site.

5.2.2 Do the websites about an issue cluster together based on ideology, type of source, or some other factor?

Research into political blogs has found that their hyperlinking patterns demonstrate homophily, the tendency for someone to associate with people or ideas similar to themselves and their ideas. In a study of blog posts leading up to the 2004 presidential election, over 90 percent of intra-blog hyperlinks were from conservative-to-conservative or liberal-to-liberal [4]. A researcher could look for for this pattern in the websites about each issue by analyzing the Wordfish score of websites and the websites to which they hyperlink. Also, a researcher could look for clustering based on the type of source or other factors.

5.3 Concluding Remarks

These and other questions can be answered by coupling the tools of the computer science field with the substance of political science research. As the use of the Internet expands, research and continued theory development on this and related topics will be critical to our understanding of the web's role and potential effect on politics.

Bibliography

[1] Cho, Junghoo and Roy, Sourashis. 2004. “Impact of search engines on page popularity.” WWW ’04: Proceedings of the 13th international conference on World Wide Web. 20-29. (New York: ACM)

[2] Assard, Erik and Bennet, Lance. 1997. Democracy and the Marketplace of Ideas. (Cambridge)

[3] Carpini, Delli. Keeter, Michael and Keeter, Scott. 1996. What Americans Know About Politics and Why It Matters. Chapter 1, “From Democratic Theory to Democratic Practice: The Case for an Informed Citizenry.” (New Haven: Yale University Press)

[4] Adamic, Lada and Glance, Natalie. 2005. “The political blogosphere and the 2004 U.S. election: Divided they blog.” `http://www.blogpulse.com/papers/2005/AdamicGlanceBlogWWW.pdf`

[5] Adamic, Lada and Huberman, Bernardo. 2000. “The Nature of Markets on the World Wide Web.” Quarterly Journal of Economic Commerce 1:512.

[6] Albert, A., Jeong, H. and Barabasi, A. 1999. “Diameter of the World Wide Web.” Nature 401:130-131.

[7] Albrecht, S. 2006. “Whose voice is heard in online deliberation? A study of participation and representation in political debates on the Internet.” Information, Communication and Society 9 (1) 62-82.

[8] Barber, B.R. 1999. “Three scenarios for the future of technology and strong democracy.” Political Science Quarterly 113, 573-590.

[9] Bimber, Bruce. 2003. Information and American Democracy. (Cambridge, UK: Cambridge University Press)

[10] Brin, S. and Page, L. 1998. “The Anatomy of a Large-Scale Hypertextual Web Search Engine”. I: Seventh International World-Wide Web Conference. April 14-18, 1998. Brisbane, Australia.

[11] Budge, I. 1997. The new challenge of direct democracy. (Cambridge, UK: Politicy Press)

[12] Chakrabarti et al. 2002. “The structure of broad topics on the web”. WWW2002. `http://www.cse.iitb.ac.in/~soumen/doc/www2002t/p338-chakrabarti.pdf`

[13] Dahl, Robert A. 1989. Democracy and its Critics. (New Haven: Yale University Press)

[14] Ding, Chris, Xiaofeng He, Parry Husbands, Hongyuan Zha and Horst Simon. 2002. “PageRank, HITS and a Unified Framework for Link Analysis.” Technical Report No. 49372. LBNL.

[15] Drezner, Daniel and Farrel, Henry. 2004. “The Power and Politics of Blogs.” `http://www.danieldrezner.com/research/blogpaperfinal.pdf`

[16] Hindman, Matthew. 2009. The Myth of Digital Democracy. (Princeton: Princeton University Press)

[17] Horrigan, John, Garrett, Kelly, and Resnick, Paul. 2004. “The Internet and Democratic Debate.” `http://www.pewInternet.org/ppf/r/141/report_display.asp>`

[18] Kleinberg, Jon. 1999. ”Authoritative sources in a hyperlinked environment” Journal of the ACM 46 (5).

[19] McCallum, Andrew. 1996. “Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering.” `http://www.cs.cmu.edu/~mccallum/bow`

[20] Mitra, A. 2001. “Marginal voices in cyberspace.” New Media and Society 3, 29-48.

[21] McCombs, M.E. and Shaw, D.L. 1993. “The Evolution of Agenda-Setting Research: Twenty-Five Years in the Marketplace of Ideas.” Journal of Communication. 43.2:58-67.

[22] Muhlberger, Peter. “Polarization of Political Attitudes and Values on the Internet”. Paper presented at the annual meeting of the International Communication Association, New Orleans Sheraton, New Orleans, LA, May 27, 2004 `http://www.allacademic.com/meta/p112523_index.html`

[23] Pass, G., Chowdhury, A. and Torgeson, C. 2006. “A picture of search.” Proceedings of the 1st international conference on Scalable information systems.

[24] Pennock et al. 2002. “Winners dont take all: Characterizing the competition for links on the Web.” Proceedings of the National Academy of Sciences. `http://modelingtheweb.com/modelingtheweb.pdf`

[25] Prior, Markus. 2005. “News vs. Entertainment: How Increasing Media Choice Widens Gaps in Political Knowledge and Turnout.” American Journal of Political Science 49.3: 577-592

[26] Proksch, Sven-Oliver and Slapin, Jonathan B. 2008. “WORDFISH: Scaling Software for Estimating Political Positions from Texts.” Version 1.2. 4 September 2008, `http://www.wordfish.org`

[27] Pew Internet and American Life Project. Trend Data. `http://www.pewinternet.org/Static-Pages/Trend-Data/Online-Activities-Daily.aspx`. Accessed Fed 26 2010.

[28] Putnam, R.D. 2000. Bowling alone: the collapse and revival of American community. Simon & Schuster.

[29] Rheingol,d H. 1993. The virtual community: Homesteading on the electronic frontier. (Boston: Addison-Wesley)

[30] Rheingold, H. 2002. Smart mobs. The next social revolution. (Cambridge: Perseus Publishing)

[31] Scheufele, Dietram A. and Tewksbury, David. 2007. "Framing, agenda-setting, and priming: The evolution of three media effects models." Journal of Communication, 57.1:9-20

[32] Schwartz, E. 1996. Netactivism: How citizens use the Internet. (Sebastopol: Songline Studies)

[33] Shedletsky, Leonard J and Aitken, Joan E. 2004. Human Communication on the Internet. (Pearson)

[34] Smith, Aaron. 2008. "The Internet's Role in Campaign 2008". `http://www.pewinternet.org/Reports/2009/6--The-Internets-Role-in-Campaign-2008.aspx`

[35] Sunstein, Cass. 2001. Republic.com. (Princeton: Princeton University Press)

[36] Tan, Yue and Weaver, David H. 2007. "Agenda-setting effects among the media, the public, and congress, 1946-2004." Journalism and Mass Communication Quarterly 84:729744.

[37] Trippi, Joe. 2005. The Revolution Will Not Be Televised: Democracy, the Internet, and the Overthrow of Everything. (Regan Books)

Appendix A

Program to connect to Rainbow server and classify a webpage

```
#!/usr/bin/python

import logging
import os
import platform
import re
import subprocess
import telnetlib
import time

class WebsterClassifier:

  next_port = 2000

  def __init__(self, category, port=None, uniform=False):
    self.logger = logging.getLogger('')
    self.category = category
    self.host = 'localhost'
    self.uniform = uniform

    if port:
      self.port = port
    else:
      self.port = self.__class__.next_port
      self.__class__.next_port += 1

    self.startup()



  def __del__(self):
    self.shutdown()

  def startup(self):

    if platform.architecture()[0] == '32bit':
      model_path = "/home/brian/thesis/bow_models/%s/" % self.category
    else:
      model_path = "/home/brian/thesis/bow_models_64/%s/" % self.category

    command = ['rainbow', '--verbosity=1', '-d', model_path, "--score-precision=2", "--skip-html", "--
        query-server=%d" % self.port]
    if self.uniform == True:
      command.append('--uniform-class-priors')
    self.logger.debug("Starting rainbow on port %d" % self.port)
    self.process = subprocess.Popen(command, stdout=open("%d.log" % self.port, 'w'), stderr=subprocess.
        STDOUT)

    self.rainbow_connection = None
    while self.rainbow_connection == None:
      try:
        self.logger.debug("Trying to connect to rainbow on %s" % self.port)
        self.rainbow_connection = telnetlib.Telnet(self.host, self.port)
      except:
        if self.process.poll() != None:
          raise WebsterClassifierError("Rainbow on %s exited with %s" % (self.port, self.process.
              returncode))
      time.sleep(10)

  def shutdown(self):
    if not self.rainbow_connection == None:
      self.rainbow_connection.close()
    self.logger.debug('Shutting down rainbow')
    if hasattr(self.process, 'terminate'):
      self.process.terminate()
    else:
      subprocess.call(['kill', "%d" % self.process.pid])
    self.process.wait()

  def classify(self, document):
    result = None
    while result == None:
      try:
        self.rainbow_connection.write(document)
        # control sequence to signal end of document
        self.rainbow_connection.write("\r\n.\r\n")
        result = self.rainbow_connection.read_some()
      except:
        self.logger.warn("Restarting rainbow on %s" % self.port)
        self.shutdown()
        self.startup()

    pattern = re.compile(r"%s (\d+\.*\d*)" % self.category)
    match = re.search(pattern, result)

    try:
      probability = match.group(1)
      score = float(probability)
      # rainbow returns really low scores in scientific notation
      # the code above will mistakenly convert those into a score greater than 1
      # this corrects for that
      if score > 1.0:
        return 0.0
      else:
        return score
    except:
      raise WebsterClassifierError("Classifying via rainbow on %s failed. Result was %s" % (self.port,
          result))

class WebsterClassifierError(Exception):
```

```
"""Exception raised for errors in communication with classifier."""
pass
```

Appendix B

Program to fetch webpages

```
#!/usr/bin/python

import focuser

import chardet
import logging
import optparse
import Pyro.core
import Queue
import re
import robotparser
import socket
import time
import urllib2
import urlparse

class Webster:
  def __init__(self, master, classifier_queue, master_lock=None, store_text=False):
    self.logger = logging.getLogger('')
    self.master = master
    self.master_lock = master_lock
    self.classifier_queue = classifier_queue
    self.store_text=store_text
    self.status = {'NOT_VISITED': 0, 'IN_PROGRESS': 1, "VISITED": 2,
      "VISIT_ERROR": 3, "CLASSIFY_ERROR": 4, "DECODE_ERROR": 5}
    self.cutoff = 1.0
    self.delay = 15
    # this should match table definition in database
    self.max_url_length = 333
    self.charset_pattern = re.compile('["\']text/html; charset=(.*?)["\']', re.IGNORECASE)
    self.link_pattern = re.compile('href=["\'].*?["\']', re.IGNORECASE)
    self.exclude_pattern = re.compile('(\.css|\.ico|\.jpg|\.png|\.gif)', re.IGNORECASE)

    urllib2.UserAgent='Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.1.6) Gecko/20100107 Fedora
        /3.5.6-1.fc12 Firefox/3.5.6'
    socket.setdefaulttimeout(self.delay)

  def page_error(self, url, error):
    """Wrapper to handle locking before submitting error"""
    if self.master_lock:
      self.master_lock.acquire()
    try:
      self.logger.debug("Submitting error %s for url %s" % (error, url))
      self.master.submit_error(url, error)
    except Exception,x:
      self.logger.error( ''.join(Pyro.util.getPyroTraceback(x)) )
    finally:
      if self.master_lock:
        self.master_lock.release()
    return True

  def page_success(self, url, relevance, unique_links=None, link_count=None, html=None):
    """Wrapper to handle locking and soem other processing before submitting page"""
    if self.master_lock:
      self.master_lock.acquire()

    if self.store_text == True:
      html = html.encode('zlib').encode('base64')

    try:
      self.logger.debug("Submitting page %s with relevance %f" % (url, relevance))
      # work around difficulty of sending None type over xmlrpc
      if relevance < self.cutoff:
        if self.store_text == False:
          self.master.submit_page(url, relevance)
        else:
          self.master.submit_page(url, relevance, html=html)
      else:
        if self.store_text == False:
          self.master.submit_page(url, relevance, unique_links, link_count)
        else:
          self.master.submit_page(url, relevance, unique_links, link_count, html)
    except Exception,x:
      self.logger.error( ''.join(Pyro.util.getPyroTraceback(x)) )
    finally:
      if self.master_lock:
        self.master_lock.release()
    return True

  def crawl_page(self):
    """Crawl a page"""
    if self.master_lock:
      self.master_lock.acquire()
    try:
      url = self.master.get_page()
      self.logger.debug("Got page %s" % url)
    except:
      self.logger.error("Could not communicate with dbserver")
      time.sleep(self.delay)
      return False
    finally:
      if self.master_lock:
        self.master_lock.release()

    if url == None:
      self.logger.warning("Crawler did not receive a page from master")
      time.sleep(self.delay)
      return False
```

```
parsed = urlparse.urlparse(url)
url = parsed.geturl()

try:
  rp = robotparser.RobotFileParser()
  rp.set_url('http://' + parsed.netloc + '/robots.txt')
  self.logger.debug("Reading robots.txt")
  rp.read()
  fetchable = rp.can_fetch(urllib2.UserAgent, url)
except:
  self.logger.debug("Processing robots.txt failed")
  self.page_error(url, self.status['VISIT_ERROR'])
  return True

if not fetchable:
  self.logger.debug("Crawler is blocked by robots.txt")
  self.page_error(url, self.status['VISIT_ERROR'])
  return True

try:
  self.logger.debug('Connecting to page')
  req = urllib2.Request(url)
  req.add_header("User-Agent", urllib2.UserAgent)
  opener = urllib2.build_opener()
  input = opener.open(req)
  opener.close()
except:
  self.logger.debug("Connecting failed")
  self.page_error(url, self.status['VISIT_ERROR'])
  return True

if input.headers.type != 'text/html':
  self.logger.debug('Page was not html')
  input.close()
  self.page_error(url, self.status['VISIT_ERROR'])
  return True

try:
  self.logger.debug("Reading html")
  raw_html = input.read()
  input.close()
  self.logger.debug("Snippet of html is %s" % raw_html[:40])
except:
  self.logger.debug("Reading html failed")
  self.page_error(url, self.status['VISIT_ERROR'])
  return True

# avoid bad erros where python assumes page is ascii but its not

charsets_to_try = []
charset_identified = False

try:
  ignore, charset_header = input.headers.getheader('content-type').split('charset=')
  charsets_to_try.append(charset_header)
  self.logger.debug("Character set specified in headers is %s" % charset_header)
```

```
except:
  self.logger.debug('No character set specified in headers')

charset_match = self.charset_pattern.search(raw_html)
if charset_match:
  charset_meta = charset_match.group(1)
  charsets_to_try.append(charset_meta)
  self.logger.debug("Character set specified in meta tag is %s" % charset_meta)
else:
  self.logger.debug('No character set specified in meta tag')

for c in charsets_to_try:
  try:
    html = raw_html.decode(c)
    charset_identified = True
    break
  except:
    self.logger.debug("Character set %s was wrong" % c )
    continue

if not charset_identified:
  self.logger.debug("Trying to detect character set on %s" % parsed.netloc)
  try:
    charset = chardet.detect(raw_html)['encoding']
    self.logger.debug("Detected character set %s" % charset)
    html = raw_html.decode(charset)
  except:
    self.logger.warn("Detected character set was wrong")
    self.page_error(url, self.status['DECODE_ERROR'])
    return True

  # convert from unicode to bytes for classifier, replacing characters that aren't valid in ascii
try:
    html_ascii = html.encode('ascii', 'xmlcharrefreplace')
except:
    self.logger.warn("Error encoding for classifier")
    self.page_error(url, self.status['DECODE_ERROR'])
    return True

try:
  self.logger.debug("Getting Classifier at %f" % time.clock())
  classifier = self.classifier_queue.get(block=True)
  self.logger.debug('Classifying')
  probability = classifier.classify(html_ascii)
except focuser.WebsterClassifierError, e:
  self.logger.warn("Classifying page %s failed with error %s" % (url, e))
  self.page_error(url, self.status['CLASSIFY_ERROR'])
  return True
finally:
  self.logger.debug("Putting Classifier at %f"  % time.clock())
  self.classifier_queue.put(classifier)

if probability < self.cutoff:
  self.logger.debug("Probability %f not relevant" % probability)
  self.page_success(url, probability)
  return True
```

```
      unique_links = {}
      link_count = {}
      self.logger.debug('finding links')
      links = self.link_pattern.findall(html)
      for href in links:
        self.logger.debug("Found link %s" % href)
        # means start after href="
        target = urlparse.urlparse(href[6:-1])
        if self.exclude_pattern.search( target.path ):
          self.logger.debug("Link was to excluded filetype")
          continue
        if target.scheme != '' and target.scheme != 'http':
          self.logger.debug("Link was not http")
          continue
        # Add domain name to relative links
        if target.netloc == '':
          try:
            target = urlparse.urlparse( urlparse.urljoin(url, target.geturl()) )
          except:
            self.logger.warn('Error processing relative link')
            continue
        self.logger.debug('Storing link')
        # limit key to maximum length we can store in database
        unique_links[target.geturl()[:self.max_url_length]] = True
        count = link_count.setdefault(target.netloc, 0)
        link_count[target.netloc] = count + 1
        self.logger.debug("Incremented link count to %d" % link_count[target.netloc])
      self.page_success(url, probability, unique_links, link_count, html)
      return True

    def crawl_pages(self, count=100):
      start_time = time.time()
      for i in range(0, count):
        did_crawl = self.crawl_page()
        if did_crawl == False:
          return False
      ppm = count / ( (time.time() - start_time) / 60.0)
      self.logger.info("A thread crawled at %f pages per minute" % ppm)
      return True

# for quick testing with one crawler
if __name__ == '__main__':
  LOGGING_LEVELS = {'critical': logging.CRITICAL,
          'error': logging.ERROR,
          'warning': logging.WARNING,
          'info': logging.INFO,
          'debug': logging.DEBUG}

  parser = optparse.OptionParser()
  parser.add_option('-p', '--page-count', type='int', default=100, help='Number of pages to crawl')
  parser.add_option('-l', '--logging-level', help='Logging level')
  parser.add_option('-s', '--server', type='str', default='localhost', help='Address of WebsterDBServer')
  (options, args) = parser.parse_args()
  category = args[0]
```

```
logging_level = LOGGING_LEVELS.get(options.logging_level, logging.NOTSET)
logging.basicConfig(level=logging_level)
logger = logging.getLogger('')

classifier_queue = Queue.Queue(1)
classifier_queue.put( focuser.WebsterClassifier(category) )
w = Webster(Pyro.core.getProxyForURI("PYROLOC://%s:4242/webster" % options.server), classifier_queue)
w.crawl_pages(options.page_count)
```

Appendix C

Program to start multiple classifiers and crawlers

```
#!/usr/bin/python

import crawler
import focuser

import logging
import optparse
import Pyro.core
import Queue
import thread
import threading


LOGGING_LEVELS = {'critical': logging.CRITICAL,
    'error': logging.ERROR,
    'warning': logging.WARNING,
    'info': logging.INFO,
    'debug': logging.DEBUG}

parser = optparse.OptionParser()
parser.add_option('-l', '--logging-level', help='Logging level')
parser.add_option('-t', '--thread-count', type='int', default=1, help='Number of threads')
parser.add_option('-c', '--classifier-count', type='int', default=1, help='Number of classifiers')
parser.add_option('-p', '--page-count', type='int', default=100, help='Number of pages to crawl before
    recreating a thread')
parser.add_option('-f', '--classifier-port', type='int', default=2000, help='Port to create first
    classifier on. Others will increment by 1.')
parser.add_option('-s', '--server', type='str', default='localhost', help='Address of WebsterDBServer')
(options, args) = parser.parse_args()

logging_level = LOGGING_LEVELS.get(options.logging_level, logging.NOTSET)
logging.basicConfig(level=logging_level)
logger = logging.getLogger('')
#formatter = logging.Formatter("%(relativeCreated)d %(thread)d - %(message)s")
#logger.setFormatter(formatter)

# yeah, classifer and focuser mean the same things

logger.info("Thread count is %d" % options.thread_count)
logger.info("Classifer count is %d" % options.classifier_count)
logger.info("Focuser port is %d" % options.classifier_port)

category = args[0]
```

```
classifier_queue = Queue.Queue(options.classifier_count)
sem = threading.Semaphore(options.thread_count)
master_lock = threading.Semaphore()

def run():
  try:
    logger.debug('Starting crawler')
    c = crawler.Webster( Pyro.core.getProxyForURI("PYROLOC://%s:4242/webster" % options.server),
         classifier_queue, master_lock=master_lock)
    c.crawl_pages(options.page_count)
  finally:
    sem.release()


for port in range(options.classifier_port, options.classifier_port + options.classifier_count):
  logger.info("Starting classifier on %s" % port)
  classifier_queue.put( focuser.WebsterClassifier(category), port=port )


while True:
  try:
    sem.acquire()
    thread.start_new_thread( run,() )
  except:
    logger.warn('Error occured in thread')
```

Appendix D

Program to store crawl results in a database

```
#!/usr/bin/python

import logging
import MySQLdb
import optparse
import Pyro.core
import random
import re
import sys
import time
import urlparse

class WebsterDBServer(Pyro.core.SynchronizedObjBase):
  def __init__(self,host,db):
    Pyro.core.SynchronizedObjBase.__init__(self)
    self.logger = logging.getLogger('')
    self.host = host
    self.db = db
    self.status = {'NOT_VISITED': 0, 'IN_PROGRESS': 1, "VISITED": 2, "VISIT_ERROR": 3, "CLASSIFY_ERROR":
        4, "DECODE_ERROR": 5}
    self.max_depth = 2
    self.current_depth = 2
    self.cutoff = 1.0
    self.retry_limit = 3
    self.retries = 0
    self.id_by_hostname = {}
    self.pages_in_progress = []
    self.received_time = 0
    self.pages_received = 0

    logger.debug('Connecting to mysql')
    self.conn = MySQLdb.connect(host=self.host, db=self.db, user='webster', passwd='fuSWuw2E', charset='
        utf8')

    def get_cursor(self):
    try:
      cursor = self.conn.cursor()
      cursor.execute('SELECT 1')
      return cursor
    except:
      self.logger.warn('Error in mysql connection. Retrying.')
      self.conn = MySQLdb.connect(host=self.host, db=self.db, user='webster', passwd='fuSWuw2E', charset='
          utf8')
      return self.get_cursor()

  def get_page(self):
    """Gets a page to process"""
    cursor = self.get_cursor()
    # crawl must run in order by depth (or id, which should also order by depth) to get correct results
    if not self.pages_in_progress:
      ppm = self.pages_received / ( (time.time() - self.received_time) / 60.0)
      self.logger.info("Crawling at %f pages per minute" % ppm)
      if not cursor.execute("SELECT id, url FROM pages WHERE status = %s AND depth = %s LIMIT 1000", (self
          .status['NOT_VISITED'], self.current_depth)):
        # if we didnt get a page because theyve all been visited, retry any with errors
        if self.retries < self.retry_limit:
          self.retries += 1
          self.logger.info("Preparing retry %d for pages at depth %d with errors" % (self.retries, self.
              current_depth))
          cursor.execute("UPDATE pages SET status = %s WHERE status = %s and depth = %s", (self.status['
              NOT_VISITED'], self.status['VISIT_ERROR'], self.current_depth))
          cursor.execute("UPDATE pages SET status = %s WHERE status = %s and depth = %s", (self.status['
              NOT_VISITED'], self.status['CLASSIFY_ERROR'], self.current_depth))
          return self.get_page()
        # if we didn't get a page because there are no more at this depth, increase depth
        elif self.current_depth < self.max_depth:
          self.current_depth += 1
          self.logger.info("Switching to depth %d" % self.current_depth)
          # will want retries once done with this depth to
          self.retries = 0
          return self.get_page()
        # if were at max depth and through with retries, log it and return nothing
        else:
          self.logger.warn('No more unvisited pages in database')
          return None
      pages_tuple = cursor.fetchall()
      self.logger.debug("Got %d new pages from database" % cursor.rowcount)
      self.pages_received = cursor.rowcount
      self.received_time = time.time()
      for id, url in pages_tuple:
        cursor.execute("UPDATE pages SET status = %s WHERE id = %s", (self.status['IN_PROGRESS'], id))
      self.pages_in_progress = list(pages_tuple)
      # put the pages in random order to avoid hammering a single site or having all crawlers stall on a
          slow site
      random.shuffle(self.pages_in_progress)
    # get url from beginning of list, then remove that entry from list
    url = self.pages_in_progress[0][1]
    self.pages_in_progress = self.pages_in_progress[1:]
    return url

  def get_site_id(self, hostname):
    """Return id of site, creating database entry if not present"""
    cursor = self.get_cursor()
    if hostname in self.id_by_hostname:
      id = self.id_by_hostname[hostname]
    else:
      if not cursor.execute("SELECT id FROM sites WHERE hostname = %s", (hostname,)):
        self.logger.debug("site %s was not in database" % hostname)
        cursor.execute("INSERT INTO sites (hostname) VALUES (%s)", (hostname,))
        cursor.execute("SELECT id FROM sites WHERE hostname = %s", (hostname,))
      id, = cursor.fetchone()
    self.logger.debug("site %s has id %d" % (hostname, id))
    return id

  def queue_page(self, url, queue_depth):
    """Places new page in the database"""
    #assert type(url) is str
    #assert type(queue_depth) is int
    cursor = self.get_cursor()
    parsed = urlparse.urlparse(url)
    url = parsed.geturl()
    if parsed.scheme != 'http':
      # this happens rarely
      self.logger.debug("Will not queue not http page %s" % url)
      return False
    if not cursor.execute("SELECT url FROM pages WHERE url = %s",(url,)):
      try:
        site_id = self.get_site_id(parsed.netloc)
      except:
        self.logger.warn("Will not queue error getting site id for %s" % parsed.netloc)
        return False
      cursor.execute("INSERT INTO pages (site_id, depth, url) VALUES (%s, %s, %s)", (site_id, queue_depth,
           url))
      self.logger.debug("Queued with depth %d page %s" % (queue_depth, url))
      return True
    else:
      self.logger.debug("Will not queue already in database page %s" % url)
      return False

  def submit_page(self, url, relevance, unique_links=None, link_count=None, html=None):
    """Updates the database with information about the page"""
    #assert type(url) is str
    #assert type(relevance) is float
    cursor = self.get_cursor()
    cursor.execute("SELECT id, site_id, depth FROM pages WHERE url=%s", (url,))
    (page_id, from_id, depth) = cursor.fetchone()
    self.logger.debug("page id %d is from %d and has depth %d" % (page_id, from_id, depth))
    cursor.execute("UPDATE pages SET status = %s, relevance = %s WHERE id = %s LIMIT 1", (self.status['
        VISITED'], relevance, page_id))
    self.logger.debug("Set visited with relevance %f page id %d" % (relevance, page_id))
    if relevance >= self.cutoff:
      #assert type(link_count) is dict
      #assert type(unique_links) is dict
      for hostname, count in link_count.iteritems():
        try:
          to_id = self.get_site_id(hostname)
        except:
          self.logger.warn("Will not insert links error getting site id for %s" % hostname)
          continue
        if not cursor.execute("SELECT count from links WHERE from_id = %s AND to_id = %s", (from_id, to_id
            )):
          self.logger.debug("%d links are first instance from %d to %d" % (count, from_id, to_id))
          cursor.execute("INSERT INTO links (from_id, to_id, count) VALUES (%s, %s, %s)", (from_id, to_id,
               count))
        else:
            old_count, = cursor.fetchone()
            self.logger.debug("Already %d links from %d to %d" % (old_count, from_id, to_id))
            new_count = old_count + count
            cursor.execute("UPDATE links SET count = %s WHERE from_id = %s AND to_id = %s LIMIT 1", (
                new_count, from_id, to_id))
            self.logger.debug("Updated to %d links from %d to %d" % (new_count, from_id, to_id))
        if not html == None:
          cursor.execute("INSERT INTO page_text (page_id, html) VALUES (%s, %s)", (page_id, html))
          self.logger.debug("Inserted html")
        if depth < self.max_depth:
          self.logger.debug("Below max depth so queuing new pages")
          for link in unique_links.iterkeys():
            try:
              self.queue_page(link, depth+1)
            except:
              self.logger.warn("Error queueing page %s" % link)
      return True

    def submit_error(self, url, error):
      """Updates the database with the error that occured when processing the page"""
      #assert type(url) is str
      #assert type(error) is int
      cursor = self.get_cursor()
      cursor.execute("UPDATE pages SET status = %s WHERE url = %s LIMIT 1", (error, url))
      self.logger.debug("Set error %d for page %s" % (error, url))
      return True


  LOGGING_LEVELS = {'critical': logging.CRITICAL,
            'error': logging.ERROR,
            'warning': logging.WARNING,
            'info': logging.INFO,
            'debug': logging.DEBUG}
  parser = optparse.OptionParser()
  parser.add_option('-l', '--logging-level', help='Logging level')
  parser.add_option('-a', '--bind-address', type='str', default='localhost', help='Address to listen on')
  parser.add_option('-p', '--bind-port', type='int', default=4242, help='Port to listen on')
  parser.add_option('-m', '--mysql-address', type='str', default='localhost', help='Address of MySQL server'
      )
  (options, args) = parser.parse_args()

  logging_level = LOGGING_LEVELS.get(options.logging_level, logging.NOTSET)
  logging.basicConfig(level=logging_level)
  logger = logging.getLogger('')
  category = args[0]

  logger.debug('Setting up server')
  Pyro.core.initServer()
  daemon=Pyro.core.Daemon(host=options.bind_address, port=options.bind_port)
  uri=daemon.connect(WebsterDBServer(host=options.mysql_address, db=category),"webster")

  try:
    logger.info("Server is listening at %s" % uri)
    daemon.requestLoop()
  except KeyboardInterrupt:
    logger.info('Server is shutting down')
```

```
daemon.shutdown(True)
```