Filtering Spam Results Out of Search Queries

Daniel Holevoet, under advisement of Dana Angluin

December 15, 2006

Abstract

While modern search engines do a reasonable job organizing search results by relevance, the lack of a major shift in the underlying technology has allowed exploitation to occur. Spam results are becoming more frequent in some subset of queries, particularly those topics which are similar to the types of topics which are considered spam in email. It is hypothesized that using the same technique that is used to filter spam email can be used to filter spam search results.

1 Introduction

1.1 The Problem

Current page ranking algorithms are designed to rank pages based on the number of times that other pages link to them. While PageRank\(^\text{[1]}\) provides significant advances over keyword-only search algorithms, it does not solve the problem of link spamming\(^\text{[2]}\), such as in comment boxes on sites and forums. Therefore, another method is needed to determine whether a search result is valid or whether it represents spam content.

As further justification that the problem of link spamming is real, the author receives dozens of link spam messages via a comment box on his website, which the link spam crawlers have mistaken as a page that updates a page on the internet. Thinking that sending comments will allow them to post urls, the robots constantly fill out and submit the comment form.

1.2 PageRank

PageRank applies standard citation analysis to web pages, considering each page as analogous to some document in a given community, and links on each page as citations to other documents. These citations lend credibility
to the pages which they reference, boosting the PageRank of those pages, and increasing their position in query results in which they are present.

However, PageRank falls flat when we consider current exploitations of the system. The ubiquity of blogs, web forums, and other sites with open (and often entirely unprotected) comment systems allows malicious surfers to artificially boost the rank of certain pages by posting comments with links on these open sites. The comments in effect siphon rank from these other pages towards the target site.

Frequently the motivation for this siphoning is to exploit a higher Page-Rank for profit. Pages which appear higher in search results are more likely to be accessed, and more likely to appear legitimate. In these cases, the site owners are exploiting technology to reach a mass market, usually one which is uninterested in the services offered (whether it is because they are illegitimate, or otherwise undesirable). We liken these search result entries to spam via email, and attempt to use similar techniques to filter them.

1.3 Example Spam Queries
Queries which result in many spam search results are among those most frequently spotted as topics in spam emails. Specifically, queries on prescription drugs, stock tips, and free software are likely to return spam results.

2 Proposed Solutions
In order to filter spam results from our queries, users must first define the types of results that they consider spam and those that they do not. Second, data must be gathered from these pages to develop the criteria for determining whether a page is spam. Finally, our algorithm must be able to look at our sample data to make assertions about subsequent pages that users may find.

2.1 User Input
Our system bases its evaluation of pages on the examples given by the user. Similar to how an email filter behaves, our system will tailor its definition of spam to how our users define it.
2.1.1 Conducting a Query

The current implementation of our system uses Google’s API for conducting searches. When a user enters a query, it is saved in a query database and then passed on to Google. The Google API returns the top ten results of the query.

When the results are returned, our system loads (in a new thread) the links returned from the query, and downloads the content. It creates a hash of the page so that later we can determine (with reasonable accuracy) whether the page has changed since we cached it, although this is not currently used in the system. The hash is created by passing the entire HTML contents of the page through the MD5 algorithm. Pages which fail to load are given a hash value of -1, which cannot result from MD5, which is used as sufficient grounds to eliminate the url from later testing.

2.1.2 Caching Pages

When we cache a page in our database, we save the link, a hash of the page, and a list of keywords from the page. The list of keywords is generated by passing the entire HTML content through a series of regular expressions to remove HTML tags, javascript, and comments, as well as simple punctuation on word boundaries. This leaves us with a reasonably clean list of words to use later\(^1\).

Queries are also saved as well, with the query string first sorted in alphabetical word order.\(^2\) Data about how users view and rank the page, in the browser, is associated with this saved query. A single result which appears in multiple queries will have unique data associated with each query (in order to make sure we can keep the query context in mind when classifying the results\(^3\)).

2.1.3 User Interface for Tagging Spam

While the pages are being cached, the search results are passed to the user interface where they are displayed, in the order Google provides them. A snippet of the page is displayed, along with the link, title of the page, and

\(^1\)See Appendix A for a sample before and after HTML page.

\(^2\)The strings ‘apple computer’ and ‘computer apple’ are considered to be the same query, unless they are actually quoted within the search query. A quoted string is treated as a single word in the alphabetization.

\(^3\)The current implementation ignores query context, but allows us to performs operations on the association in the future.
options to view the page either inline (through an AJAX\textsuperscript{4}) or in a new window\textsuperscript{5}.

When a link is clicked, either to view the result inline, or in a new window, the internal count for the number of times that link has been clicked is incremented. Simultaneously, options are presented to rank the page in one of three categories: “Exactly what I was looking for,” “Not quite right,” and “Spam.” Currently, the first two options are grouped together as ‘non-spam’ internally. When a rank is selected and submitted, the ranking options are once again hidden.

2.2 Parsing Data

Once we’ve collected sufficient data from the user, we need to massage it into something that can be used to formulate predictions.

2.2.1 Preparations and Keyword Analysis

Our initial two spam filters consist of a pseudo-Bayes classifier and a true implementation of Naive Bayes\textsuperscript{3}. The pseudo-Bayes is implemented following the code proposed by Daniel Shiffman\textsuperscript{4}, though converted into Ruby with some modifications. The true Naive Bayes algorithm is similar, but uses an unmodified Bayes algorithm to find the probability that a page is spam.

To prepare for training our filter, we mark all of the urls in our database as having “non-training” status. Later we will mark some of these as used in training so that we do not use them for determining the accuracy of our filter. We also mark urls that return a 404, or other HTML error code, as invalid so that they are ignored for the purposes of our analysis.

Once we have prepared our database for keyword analysis, we begin by selecting all the urls that have been cached that also meet the following criteria:

- Has a hash that isn’t equal to \(-1\textsuperscript{6}\)
- Has a non-empty list of keywords
- Has at least a single user-generated ranking (clicking a link does not suffice)

\textsuperscript{4}AJAX stands for Asynchronous JavaScript and XML, a technique which allows us to update the search results page without requiring that we reload the entire page.
\textsuperscript{5}See Appendix B for example screenshots.
\textsuperscript{6}As noted in 2.1.1, \(-1\) represents an invalid page.
This list of urls is then randomly split into two segments: training urls (these urls are subsequently marked in the database) and non-training urls. We then iterate through the list of training urls.

We begin by calculating data for the pseudo-Bayes algorithm. For each url we iterate through each of the keywords and for urls that have a non-spam rank, we increment the count for that keyword, in a global table, of non-spam (“good”) occurrences. For each appearance of the word in a spam url we increment the spam (“bad”) occurrences. Once we have finished the iteration, we calculate some statistics for each keyword as follows:

\[
pSpam = \frac{\text{bad occurrences}}{\text{good occurrences} + \text{bad occurrences}}
\]

\[
\text{interesting} = |0.5 - pSpam|
\]

Further, we check the value of \(pSpam\), ensuring that it is between 0.01 and 0.99, and set it to either the lower or upper bound if it is outside of the respective end of the range. Keywords are more interesting when there is a greater disparity between the number of good and bad occurrences. We have little interest in words that are equally likely to be spam as not spam.

Then the data for Naive Bayes is calculated as follows. We iterate through the keywords again, but this time consider each unique keyword for each url, rather than each occurrence of each keyword. If a keyword in any given url appears more than once, it is only counted a single time for that url. For each appearance in a spam url, we increment the number of spam (“bad”) pages in which that keyword has appeared. Likewise, for each appearance in a non-spam url, we increment the number of non-spam (“good”) pages.

\[
p = \frac{\text{bad occurrences}}{\text{number of spam urls}}
\]

\[
q = \frac{\text{good occurrences}}{\text{number of non-spam urls}}
\]

After these values are calculated, we save the table of keywords to the database for later use.\(^7\)

\(^7\)Despite the fact that we go through two iterations of the keywords, once for each
2.2.2 Performing Classification

Once we have trained the system with the test data, we can proceed to classify the remaining pages, those in the non-training urls set. Our classification proceeds first with pseudo-Bayes classification, then with real Naive Bayes classification.

In pseudo-Bayes classification, for each of the unclassified pages we iterate through each keyword that appears in the page, saving that keyword in a list for the page and associating with it the values $p_{Spam}$ and $interesting$. When we have iterated through all the keywords for a page, we sort the list in descending order of $interesting$, and then select the first fifteen words (the most interesting fifteen). Words which appear in unclassified pages but not in our training urls are assigned a default $p_{Spam}$ of 0.4$^8$.

The $p_{Spam}$ value of these words are then combined, using an approximation of Naive Bayes (hence its reference as the pseudo-Bayes classification). We begin with two quantities:

\[
pposProduct = 1.0 \\
pnegProduct = 1.0
\]

Then, we iterate through each word modifying the products as follows:

\[
pposProduct = pposProduct \times p_{Spam} \\
pnegProduct = pnegProduct \times (1.0 - p_{Spam})
\]

Now the final probability can be determined:

\[
combinedProb = \frac{pposProduct}{pposProduct + pnegProduct}
\]

If $combinedProb \geq threshold$ the page is classified as spam. This is then compared to the page’s actual ranking to determine whether the classification is accurate.

$^8$However, this only applies if we have less than fifteen words that are present in both the training urls and the given unclassified url. This is unlikely, but still possible.
In our real Naive Bayes classification, we consider \( p \) and \( q \) to determine the likelihood that our non-training urls are spam. For each of the unclassified pages we iterate through each keyword in the page (counting each occurrence only once) combining \( p_n \) and \( q_n \) in the following formula\(^3\):

\[
\text{combinedProb} = \frac{p_1p_2\cdots p_n}{p_1p_2\cdots p_n + q_1q_2\cdots q_n}
\]

As with the pseudo-Bayes algorithm, if \( \text{combinedProb} \geq \text{threshold} \) the page is classified as spam. Then the classification is compared with the page’s human classification.

### 2.2.3 Sample Data, Parameters, and Accuracy

The sample data consists of 148 manually-classified pages, of which 98 are non-spam and 50 are spam.

Both pseudo-Bayes and Naive Bayes are customizable, with a number of ways to change the way the algorithm behaves. The most important parameters are the number of interesting words to consider (in pseudo-Bayes) and the spam threshold (in both), above which we consider pages to be spam. However, to illustrate the major differences between the algorithms, while keeping parameters as close as possible, the results are only demonstrated with the default values\(^9\).

For each test, the system is trained by randomly selecting 74 urls (training urls). Next, the pseudo-Bayes and Naive Bayes classifiers are compared by classifying the other 74 urls (the same 74 for both pseudo-Bayes and Naive Bayes). The first table represents the number of spam urls that happened to be in each test set.

The second table shows the results of testing with the pseudo-Bayes algorithm. The third table shows the results of testing with the Naive Bayes algorithm. Accuracy is determined by taking the total number of correct classifications in each set over the total number of possible correct classifications.

Tagged spam and tagged non-spam are counts of correctly tagged spam and non-spam pages. False positives and false negatives represent pages that should have been marked as non-spam and spam respectively, but were classified in the opposite manner.

\(^9\)The number of interesting words defaults to fifteen, and the spam threshold defaults to 90%.
<table>
<thead>
<tr>
<th></th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
<th>Test 4</th>
<th>Test 5</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td># of spam urls</td>
<td>25</td>
<td>23</td>
<td>26</td>
<td>22</td>
<td>25</td>
<td>24.2</td>
</tr>
<tr>
<td># of total urls</td>
<td>74</td>
<td>74</td>
<td>74</td>
<td>74</td>
<td>74</td>
<td>74</td>
</tr>
</tbody>
</table>

### Pseudo-Bayes

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>74%</td>
<td>76%</td>
<td>81%</td>
<td>65%</td>
<td>74%</td>
<td>74%</td>
</tr>
<tr>
<td>Tagged Spam</td>
<td>7</td>
<td>12</td>
<td>11</td>
<td>4</td>
<td>8</td>
<td>8.4</td>
</tr>
<tr>
<td>False Positives</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1.8</td>
</tr>
<tr>
<td>Tagged Non-Spam</td>
<td>48</td>
<td>44</td>
<td>49</td>
<td>44</td>
<td>47</td>
<td>46.4</td>
</tr>
<tr>
<td>False Negatives</td>
<td>18</td>
<td>15</td>
<td>13</td>
<td>24</td>
<td>17</td>
<td>17.4</td>
</tr>
</tbody>
</table>

### Naive Bayes

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>76%</td>
<td>72%</td>
<td>73%</td>
<td>62%</td>
<td>74%</td>
<td>71.4%</td>
</tr>
<tr>
<td>Tagged Spam</td>
<td>14</td>
<td>15</td>
<td>14</td>
<td>6</td>
<td>13</td>
<td>12.4</td>
</tr>
<tr>
<td>False Positives</td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>7.8</td>
</tr>
<tr>
<td>Tagged Non-Spam</td>
<td>42</td>
<td>38</td>
<td>40</td>
<td>40</td>
<td>42</td>
<td>40.4</td>
</tr>
<tr>
<td>False Negatives</td>
<td>11</td>
<td>12</td>
<td>10</td>
<td>22</td>
<td>12</td>
<td>13.4</td>
</tr>
</tbody>
</table>

Overall, the results demonstrate that the pseudo-Bayes algorithm is more accurate, however, both algorithms succeed in different areas. Pseudo-Bayes is preferable if we’re trying to tag pages with as few false positives as possible, perhaps at the cost of more spam pages left untagged. But, in a scenario where we prefer to mark significantly more spam pages, at the cost of more false positives, then the true Naive Bayes algorithm is more successful.

It is likely that both algorithms would benefit from more training data. In particular, some pages in the real Naive Bayes algorithm were marked with a 100% probability of being spam (despite the fact that they were not spam), likely because many of the words on the page were not found in training data, and the few keywords that were found were marked as likely spam words.

### 2.3 Caveats

There are some caveats to the training and classification process.

First, as mentioned above, the training is from a relatively limited sample set of pages. This is due to the constraints of a single individual (the author) who is responsible for the training classifications.

Second, and directly related to the first caveat, is that the manual classifications are subjective, and the classifiers will attempt to classify pages into the author’s definition of spam and non-spam.
Third, the context of queries is not taken into account when doing the analysis, so pages that have likely spam words are penalized even if those words appear in the query (and are thus ranked highly in the Google search).

2.4 Implementation Notes

Our system is implemented in Ruby on Rails, with a MySQL database as a backend. Direct interaction with users is accomplished via a browser from pages generated by Rails. Developer-level interaction with data (for analysis) is accomplished via the Rails console, which allows manipulation of the database and application calls through a terminal.

3 Conclusion and Next Steps

Our results show that application of pseudo-Bayes and Naive Bayes are viable methods to filter out spam pages from legitimate pages. However, there are some clear paths laid out that would improve this system.

First, although the system succeeds from a statistical perspective, the results of the classification are not yet tied back into the user interface. A logical next step would be to allow classification of spam pages before they are presented in search results.

Second, the search context is ignored in our classification, a caveat mentioned above. Taking the search context into consideration would likely improve the accuracy of the classifiers.

Third, it may be possible to further classify spam pages into clusters rather than view them as a homogenous group. In the future, the user interface could allow a mechanism for the user to identify clusters of spam pages (those would be particular sets of spam pages which are all about similar topics) and have the classifier use this data to decide not only if the page is spam, or not, but whether it belongs in a specific spam cluster.

A Sample Data

The following is an example of a sample page (from exitthree.com), first in raw HTML form, and then after it has been stripped of HTML, leaving a list of keywords.

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```
<html xmlns="http://www.w3.org/1999/xhtml"> <head> <title>Exitthree | Home</title> <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" /> <link href="/css/main.css" rel="stylesheet" type="text/css" /> <script src="/javascripts/prototype.js" type="text/javascript"></script> </head> <body class="main"> <div id="logo"> <img src="/images/logo.gif" width="497" height="53" alt="Exitthree.com" /> </div> <div id="nav"> <ul> <li class="first"> &gt;&gt;&gt;</li> <li><a href="/">Home</a></li> <li><a href="/portfolio">Portfolio</a></li> <li><a href="/portfolio/development">In Development</a></li> <li><a href="/links">Links</a></li> <li><a href="/contact">Contact Us</a></li> <li class="last">&nbsp;</li> </ul> </div> <div id="content"> <div id="main"> <h2>Welcome to Exitthree.com</h2> <p>According to A List Apart, Good Designers Redesign, Great Designers Realign. As such, I would like to welcome visitors to the newly realigned Exitthree.com, now developed using Ruby on Rails.</p> </div> <div id="copyright"> <p>All original content copyright © 2003-2005 by Daniel Holevoet. Exitthree and Exitthree.com are trademarks of Daniel Holevoet. All rights reserved.</p> </div> </div> </body> </html>

exitthree home home portfolio in development links contact us welcome to exitthree.com according to a list apart good designers redesign great designers realign as such i would like to welcome visitors to the newly realigned exitthree.com now developed using ruby on rails all original content copyright 2003-2005 by daniel holevoet exitthree and exitthree.com are trademarks of daniel holevoet all rights reserved

B User Interface

The following are example screenshots of the user interface in the browser. The first image is the results page from a sample query. The second image is one of the results that has been expanded with the inline page view and that is presenting the ranking options. The third image is a close-up of the ranking options.
Looking for something?

daniel holevoet

Saved as: daniel holevoet

1. Daniel Holevoet & Sarah Price | New Window
   http://zoo.cs.yale.edu/classes/cs457/digital%20Copyright.pdf
   Copyright Reforms for the Digital Age: A Closer Look at Google. Daniel Holevoet &
   Sarah Price. Page 2. Introduction. What We Cover: History of copyright ...  

2. Copyright Reforms for the Digital Age: A Closer Look at Google | New Window
   http://zoo.cs.yale.edu/classes/cs457/Copyright%20Reforms%20for%20the%20Digital%20Age.pdf
   Copyright Reforms for the Digital Age: A Closer Look at Google. Daniel Holevoet
   and Sarah Price. Page 2. 1. When one of the authors discovered that her ...  

3. Linux Online - News by Daniel Holevoet | New Window
   News by Daniel Holevoet. Advertisement. - Tables turn in campus Mac vs. Windows feud,
   Feb 25 2005. Education In Yale's Computer Science Department and CS ...  

4. Exitthree | Portfolio | New Window
   http://www.exitthree.com/porfolio
   Exitthree.com serves as a portfolio for the work of Daniel Holevoet. ...  
   Exitthree and Exitthree.com are trademarks of Daniel Holevoet. All rights reserved.  

5. Exitthree | Home | New Window
   http://www.exitthree.com/
   All original content copyright ©2003-2005 by Daniel Holevoet. Exitthree and
   Exitthree.com are trademarks of Daniel Holevoet. All rights reserved.

3. Hide

Linux Online - News by Daniel Holevoet | New Window


News by Daniel Holevoet. Advertisement. - Tables turn in campus Mac vs. Windows feud,
Feb 25 2005. Education In Yale's Computer Science Department and CS ...  

Linux Online - News by Daniel Holevoet | New Window


News by Daniel Holevoet. Advertisement. - Tables turn in campus Mac vs. Windows feud,
Feb 25 2005. Education In Yale's Computer Science Department and CS ...
References


