Filtering Spam Results Out of Search Queries

Daniel Holevoet, under advisement of Dana Angluin

May 7, 2007

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. In addition, utilization of some of the unique characteristics of how spam search results are presented, and grouped, might provide additional classification opportunities.

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[1] provides significant advances over keyword-only search algorithms, it does not solve the problem of link spamming[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 PageRank 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.

Queries are also saved, with the query string first sorted in alphabetical word order. 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).

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 perform operations on the association in the future.
options to view the page either inline (through an AJAX\(^4\)) or in a new window\(^5\). The page snippet is also used to provide an approximate likelihood that the page is spam\(^6\).

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,” as well as grouping the result into one of five categories, for clustering analysis\(^7\). 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 for Bayes and Pseudo-Bayes

Our initial two spam filters consist of a pseudo-Bayes classifier and a true implementation of Naive Bayes\(^3\). The pseudo-Bayes is implemented following the code proposed by Daniel Shiffman\(^4\), 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:

\(^4\)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.

\(^5\)See Appendix B for example screenshots.

\(^6\)This is discussed in detail in 2.2.3

\(^7\)This is discussed in detail in 2.2.4
• Has a hash that isn’t equal to -1

• Has a non-empty list of keywords

• Has at least a single user-generated ranking (clicking a link does not suffice)

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:

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

\[
\text{interesting} = |0.5 - p_{\text{Spam}}|
\]

Further, we check the value of \( p_{\text{Spam}} \), 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. This gives us two ratios for each url defined as follows:

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

As noted in 2.1.1, -1 represents an invalid page.
\[ q = \frac{\text{good occurrences}}{\text{number of non-spam urls}} \]

However, for the purposes of optimization, a \textit{metrics} table is defined to store the number of spam and non-spam urls for all pages (the denominator in the expression of \( p \) and \( q \) above), and \( p \) and \( q \) are calculated at the time of classification.

After these values are calculated, we save the table of keywords to the database for later use\(^9\).

\subsection*{2.2.2 Performing Classification for Bayes and Pseudo-Bayes}

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 \( p_{\text{Spam}} \) and \( \text{interesting} \). When we have iterated through all the keywords for a page, we sort the list in descending order of \( \text{interesting} \), 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 \( p_{\text{Spam}} \) of 0.4\(^{10}\).

The \( p_{\text{Spam}} \) 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:

\[
\begin{align*}
pposProduct &= 1.0 \\
\text{pnegProduct} &= 1.0
\end{align*}
\]

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

\(^9\)Despite the fact that we go through two iterations of the keywords, once for each algorithm, we are saving the data to a keyword table in memory before writing it once to our database. This is a performance optimization, because the bulk of time spent in this process is spent in database insertion.

\(^{10}\)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.
\[ p_{\text{pos Prod}} = p_{\text{pos Prod}} \times p_{\text{Spam}} \]
\[ p_{\text{neg Prod}} = p_{\text{neg Prod}} \times (1.0 - p_{\text{Spam}}) \]

Now the final probability can be determined:

\[ \text{combinedProb} = \frac{p_{\text{pos Prod}}}{p_{\text{pos Prod}} + p_{\text{neg Prod}}} \]

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

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\[3\]:

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

As with the pseudo-Bayes algorithm, if \( \text{combinedProb} \geq \text{threshold} \) the page is classified as spam\[11\]. Then the classification is compared with the page’s human classification.

2.2.3 Sample Data, Parameters, and Accuracy for Bayes and Pseudo-Bayes

The first set of sample data consists of 148 manually-classified pages, of which 98 are non-spam and 50 are spam. The second set of sample data consists of 249 manually-classified pages, of which 159 are non-spam and 90 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.

\[11\]It is interesting to note that unless \( \text{combinedProb} \) is calculated with accuracy greater than Ruby’s default floating point, many of the results will either round up to one, or down to zero. For this reason, the classifier uses arbitrary-length floating point for all Bayes classifications.
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\textsuperscript{12}.

For each test, the system is trained by randomly selecting 74(124) urls (training urls). Next, the psuedo-Bayes and Naive Bayes classifiers are compared by classifying the other 74(124) urls (the same 74(124) 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.

First Set

<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>74%</th>
<th>76%</th>
<th>81%</th>
<th>65%</th>
<th>74%</th>
<th>74%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagged Spam</td>
<td></td>
<td>7</td>
<td>12</td>
<td>11</td>
<td>6</td>
<td>74</td>
<td>74</td>
</tr>
<tr>
<td>False Positives</td>
<td></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></td>
<td>48</td>
<td>44</td>
<td>49</td>
<td>47</td>
<td>63</td>
<td>46.4</td>
</tr>
<tr>
<td>False Negatives</td>
<td></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>76%</th>
<th>72%</th>
<th>73%</th>
<th>62%</th>
<th>74%</th>
<th>71.4%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagged Spam</td>
<td></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></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></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></td>
<td>11</td>
<td>12</td>
<td>10</td>
<td>22</td>
<td>12</td>
<td>13.4</td>
</tr>
</tbody>
</table>

\textsuperscript{12}The number of interesting words defaults to fifteen, and the spam threshold defaults to 90%.
Second Set

<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>42</td>
<td>47</td>
<td>42</td>
<td>42</td>
<td>41</td>
<td>24.2</td>
</tr>
<tr>
<td># of total urls</td>
<td>124</td>
<td>124</td>
<td>124</td>
<td>124</td>
<td>124</td>
<td>124</td>
</tr>
</tbody>
</table>

Pseudo-Bayes

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>69%</th>
<th>73%</th>
<th>69%</th>
<th>70%</th>
<th>62%</th>
<th>68.6%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagged Spam</td>
<td></td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>11</td>
<td>6</td>
<td>8.8</td>
</tr>
<tr>
<td>False Positives</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Tagged Non-Spam</td>
<td></td>
<td>77</td>
<td>82</td>
<td>76</td>
<td>76</td>
<td>71</td>
<td>76.4</td>
</tr>
<tr>
<td>False Negatives</td>
<td></td>
<td>39</td>
<td>33</td>
<td>38</td>
<td>37</td>
<td>43</td>
<td>38</td>
</tr>
</tbody>
</table>

Naive Bayes

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>69%</th>
<th>75%</th>
<th>72%</th>
<th>77%</th>
<th>70%</th>
<th>72.6%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagged Spam</td>
<td></td>
<td>19</td>
<td>11</td>
<td>18</td>
<td>28</td>
<td>27</td>
<td>20.6</td>
</tr>
<tr>
<td>False Positives</td>
<td></td>
<td>10</td>
<td>0</td>
<td>5</td>
<td>9</td>
<td>15</td>
<td>7.8</td>
</tr>
<tr>
<td>Tagged Non-Spam</td>
<td></td>
<td>67</td>
<td>82</td>
<td>72</td>
<td>68</td>
<td>60</td>
<td>69.8</td>
</tr>
<tr>
<td>False Negatives</td>
<td></td>
<td>29</td>
<td>31</td>
<td>29</td>
<td>20</td>
<td>22</td>
<td>26.1</td>
</tr>
</tbody>
</table>

In the first testing set, pseudo-Bayes performs better than Naive Bayes, suggesting that either pseudo-Bayes works better when there is a higher ratio of spam/non-spam pages, or when there are fewer training pages overall. In the second testing set, Naive Bayes takes a slight edge, suggesting that it performs better when fewer spam pages are present.

A possible explanation the relative performance differences could be in the fact that pseudo-Bayes derives its answer from more interesting words, and that words become less interesting as they are diluted with the input of many more pages. Naive Bayes suffers from the opposite problem – when fewer words are in the training database, the few that are present skew the classification in one direction or another.

Another thing to note is that pseudo-Bayes had significantly fewer false positives than Naive Bayes, regardless of overall accuracy. In a system where it would be critical not to falsely tag pages as spam, pseudo-Bayes would be superior. However, whether or not pseudo-Bayes always has fewer false positives will be discussed later, after more data is presented.
2.2.4 Preparations and Keyword Analysis for Clustered Bayes

A third classification algorithm attempts to individually rank pages based on clusters of pages with similar content. This is an attempt to prevent dilution of the keyword pool by only modifying the particular counts of each keyword when that keyword appears in a page in a given cluster. Separate counts and ranks are maintained for each of the given clusters.

During the ranking process the user is prompted to rank a page into one of five clusters: casinos, concerned with online gambling, online poker, and downloadable gambling related spyware; stocks, concerned with penny stocks, fraudulent stock information, and dubious investment communities; drugs, concerned with cheap foreign pharmaceuticals and uninformative drug-specific pages (frequently related to the categories of weight loss, male enhancement, and anti-anxiety drugs); warez, concerned with free and illicit software and media downloads\(^{13}\); and knockoffs, concerned with legitimate and illegitimate replicas of name brand jewelry, watches, clothing and other expensive items.

Preparation for Clustered Bayes training proceeds similarly to non-Clustered Bayes, marking all urls as “non-training” and invalidating urls that return error codes.

Then, once for each cluster, we select all the urls that have been cached that meet the following criteria:

- Has a hash that isn’t equal to -1\(^{14}\)
- Has a non-empty list of keywords
- Has at least a single user-generated ranking (clicking a link does not suffice)
- Has a cluster association with our given cluster

This list is then randomly split into two segments: training urls and non-training urls in exactly the same manner as non-Clustered Bayes training. Next, we iterate through the training urls calculating \(p\) and \(q\) for each keyword as in Naive Bayes training, except that instead of saving these values in the existing keywords table, we save them in a special table for cluster keywords, and associate each word with the cluster in which it was found.

\(^{13}\)It is important to note that sites which actually offered free media sans spyware, were not classified as spam.

\(^{14}\)As noted in 2.1.1, -1 represents an invalid page.
2.2.5 Performing Classification for Clustered Bayes

Classification for Clustered Bayes has four modes. The first mode classifies each non-training url within its cluster. The second mode classifies each non-training url without any supplied knowledge of which cluster to which each url belongs. The third mode performs as the first, except that it uses only the $n$ most interesting words\footnote{This defaults to 15 words.}. The fourth mode uses the $n$ interesting words most found in spam pages, and the $m$ interesting words most found in non-spam pages\footnote{These also default to 15 words.}.

The first mode behaves exactly the same as Naive Bayes classification, but within the smaller domain of each cluster. It is as though we have run the classification four times, once for each cluster, training the algorithm with the urls given a rank and that cluster’s association, and then classifying the remaining urls against that training data set.

The second mode differs in that it no longer associates the non-training urls with their cluster, because assumed knowledge of which cluster to which a random url would belong is not reasonable outside of training data. This mode gives results closer to real-world performance, whereas the first mode is more of an academic benchmark.

In the second mode, each non-training url is classified with Naive Bayes once for each cluster, which generates a $p_{Spam}$ for each of the rankings. Then the algorithm takes the highest $p_{Spam}$ and uses that as the determining factor in whether or not the given page is spam. The rationale behind this decision is that a page is more likely to be non-spam than spam because there are many more words which are not associated with spam than those associated with spam. Therefore, it would seem unlikely that a page would be marked as spam within a given cluster only if it actually is a spam page within that cluster.

In the third and fourth mode, we operate under a definition of interesting which attempts to emulate that of the pseudo-Bayes classifier. How we calculate how interesting a word is, however, must differ based on how we calculate probability that a page is spam – in pseudo-Bayes, each word has a probability, in Naive Bayes, each word only has a number of good and bad appearances. Our working definition of interesting for the purposes of modes three and four has three overall meanings. The interesting words that appear most on spam pages are those words that have the greatest bad occurrences - good occurrences. Similarly, the interesting words that appear most on non-spam pages are those words that have the greatest good occurrences -
bad occurrences. Words that are just interesting with no preference to spam or non-spam appearances are those that have the largest magnitude of good occurrences - bad occurrences.

2.2.6 Sample Data, Parameters, and Accuracy for Clustered Bayes

The set of sample data consists of 230 manually-classified pages, of which 126 are non-spam and 104 are spam.

As with previous testing, the algorithms (pseudo-Bayes, Naive Bayes) are all run with their default values\(^\text{17}\). Clustered Bayes modes one and two run with their default value. Mode four runs with a range of values for \(n\) and \(m\), with the best shown.

For each test, the system is trained as before, selecting half of the total valid urls for training and classifying the other half during testing. 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. The fourth table shows the result of testing with the first mode of the Clustered Bayes algorithm (associating test urls with their clusters). The fifth table shows the result of testing with the second mode of the Clustered Bayes algorithm (no cluster association). The sixth table shows the result of testing with the fourth mode of the Clustered Bayes algorithm, with the best performing set of parameters for \(n\) and \(m\) shown.

\(^{17}\)As before, 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>49</td>
<td>51</td>
<td>50</td>
<td>52</td>
<td>55</td>
<td>51.4</td>
</tr>
<tr>
<td># of total urls</td>
<td>115</td>
<td>115</td>
<td>115</td>
<td>115</td>
<td>115</td>
<td>115</td>
</tr>
</tbody>
</table>

**Pseudo-Bayes**

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>63%</th>
<th>69%</th>
<th>65%</th>
<th>65%</th>
<th>70%</th>
<th>66.4%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagged Spam</td>
<td></td>
<td>12</td>
<td>20</td>
<td>15</td>
<td>13</td>
<td>17</td>
<td>15.4</td>
</tr>
<tr>
<td>False Positives</td>
<td></td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.4</td>
</tr>
<tr>
<td>Tagged Non-Spam</td>
<td></td>
<td>60</td>
<td>59</td>
<td>60</td>
<td>62</td>
<td>64</td>
<td>61</td>
</tr>
<tr>
<td>False Negatives</td>
<td></td>
<td>43</td>
<td>33</td>
<td>39</td>
<td>39</td>
<td>32</td>
<td>37.2</td>
</tr>
</tbody>
</table>

**Naive Bayes**

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>66%</th>
<th>71%</th>
<th>74%</th>
<th>69%</th>
<th>69%</th>
<th>69.8%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagged Spam</td>
<td></td>
<td>16</td>
<td>21</td>
<td>25</td>
<td>17</td>
<td>14</td>
<td>18.6</td>
</tr>
<tr>
<td>False Positives</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.8</td>
</tr>
<tr>
<td>Tagged Non-Spam</td>
<td></td>
<td>60</td>
<td>61</td>
<td>60</td>
<td>62</td>
<td>65</td>
<td>61.6</td>
</tr>
<tr>
<td>False Negatives</td>
<td></td>
<td>39</td>
<td>32</td>
<td>29</td>
<td>35</td>
<td>35</td>
<td>34</td>
</tr>
</tbody>
</table>

**Cluster Mode One**

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>47%</th>
<th>45%</th>
<th>48%</th>
<th>46%</th>
<th>43%</th>
<th>45.80%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagged Spam</td>
<td></td>
<td>54</td>
<td>52</td>
<td>54</td>
<td>52</td>
<td>49</td>
<td>52.2</td>
</tr>
<tr>
<td>False Positives</td>
<td></td>
<td>60</td>
<td>62</td>
<td>60</td>
<td>62</td>
<td>66</td>
<td>62</td>
</tr>
<tr>
<td>Tagged Non-Spam</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>False Negatives</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.4</td>
</tr>
</tbody>
</table>

**Cluster Mode Two**

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>47%</th>
<th>45%</th>
<th>48%</th>
<th>46%</th>
<th>43%</th>
<th>45.80%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagged Spam</td>
<td></td>
<td>54</td>
<td>52</td>
<td>54</td>
<td>52</td>
<td>49</td>
<td>52.2</td>
</tr>
<tr>
<td>False Positives</td>
<td></td>
<td>60</td>
<td>62</td>
<td>60</td>
<td>62</td>
<td>66</td>
<td>62</td>
</tr>
<tr>
<td>Tagged Non-Spam</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>False Negatives</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.4</td>
</tr>
</tbody>
</table>

**Cluster Mode Four (best)**

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>58%</th>
<th>66%</th>
<th>63%</th>
<th>62%</th>
<th>53%</th>
<th>60.40%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagged Spam</td>
<td></td>
<td>11</td>
<td>19</td>
<td>23</td>
<td>12</td>
<td>45</td>
<td>22</td>
</tr>
<tr>
<td>False Positives</td>
<td></td>
<td>4</td>
<td>5</td>
<td>11</td>
<td>4</td>
<td>50</td>
<td>14.8</td>
</tr>
<tr>
<td>Tagged Non-Spam</td>
<td></td>
<td>56</td>
<td>57</td>
<td>50</td>
<td>59</td>
<td>16</td>
<td>47.6</td>
</tr>
<tr>
<td>False Negatives</td>
<td></td>
<td>44</td>
<td>34</td>
<td>31</td>
<td>40</td>
<td>4</td>
<td>30.6</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>m</td>
<td></td>
<td>10</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>N/A</td>
</tr>
</tbody>
</table>
The sample data in these tests is unrelated to the sample data acquired for the first and second rounds of tests shown previously – this data was collected separately with the concept of clustered data in mind. With that noted, it would seem that Naive Bayes is superior to pseudo-Bayes, having both a higher accuracy and no longer flagging many false positives as it did before. This suggests that perhaps Naive Bayes benefits from such rigorously defined sets of data\textsuperscript{18}.

Unfortunately, these results show that Clustered Bayes in mode one and two is not at all viable. Neither returned any meaningful results. However, Clustered Bayes in mode four seems to hint that a salvageable version of Clustered Bayes might exist with a system to properly set $n$ and $m$ for a given set of training data.

What is odd, however, is that in mode four, four of the five best performing samples chose only interesting “non-spam” words in the analysis. The most reasonable explanation for this behavior (which was selected from among 12 variations with different balances of $n$ and $m$) is that the most interesting “spam” words are so biased towards marking pages as spam that their inclusion is enough to bias the filter, entirely. It is likely that a larger sample of pages would help to counter this bias.

It is notable that Clustered Bayes in mode one and two performed identically. The reason for this is that the number of pages tagged as spam in mode one is the minimum number of pages that can be tagged as spam in mode two – classification of each page in mode two is guaranteed to classify every page, marked as spam in mode one, as spam in mode two because each page will be required to train against the cluster to which it belongs (and which subsequently flagged the page). The fact that mode one performed so poorly gave mode two (which would, in theory, perform worse because it is acting with less information) little room to provide interesting results.

Cluster Mode three, which is not shown above, returned no meaningful results, similarly to modes one and two. But, in contrast, it did so by marking all the pages as non-spam.

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.

\textsuperscript{18}The data collected in the earlier tests was more haphazardly collected from a much wider variety of spam and non-spam pages.
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). This is even more true when categorizing pages into clusters. Frequently, the topic of the cluster will be a spam topic (in order to find the most relevant results), so many of the non-spam pages are penalized, as demonstrated in the Clustered Bayes sample results.

2.4 Live Bayes Ranking

In addition to the offline classifiers, another classifier has been implemented to attempt realtime ranking of pages while browsing. It ties directly into the user interface and ranks pages with training data that has already been processed and the Naive Bayes classifier. However, instead of ranking all of the words in a given page (which for performance reasons is not possible), it uses the words in the Google “snippet” which is returned along with the query results. While not nearly as accurate as the offline classifiers (it usually does not flag pages as spam), if a page is classified as spam by the live classifier, it is almost always spam.

To go along with this feature, users may also choose to update the training data with additional rankings without requiring retraining of the database. Coupled together, this feature, along with live ranking allows a user to watch the classifier go from untrained to one that will pick out possible spam pages in a given search topic or query.

2.5 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 help filter out spam pages from legitimate pages. Clustered
Bayes, as it stands now, is not a viable improvement to these methods, but holds some promise with future development.

In order for Clustered Bayes to succeed, there needs to be a way to rectify the problem presented by an overabundance of spam pages in each cluster. Whether this is accomplished by setting $n$ and $m$ based on an algorithm which uses the training data, or by changing the way that clustered pages are considered by the engine is unclear.

In addition, our system needs some way to decide whether a page belongs to a certain cluster, or not, aside from a spam ranking.

Finally, it may be possible to improve the accuracy of Clustered Bayes by accounting for the search query that returns the given spam urls. This would likely improve the accuracy of the classifiers for ranking non-spam pages (as they might not be penalized for containing spam words in the query).

## 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;</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</p>
<a href="http://www.alistapart.com">A List Apart</a>,
```
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.
References


