Blog Mining for the Fortune 500

James Geller, Sapankumar Parikh, and Sriram Krishnan

College of Computing Sciences,
Department of Computer Sciences,
New Jersey Institute of Technology
Newark, NJ 07102

Abstract. In recent years there has been a tremendous increase in the number of users maintaining online blogs on the Internet. Companies, in particular, have become aware of this medium of communication and have taken a keen interest in what is being said about them through such personal blogs. This has given rise to a new field of research directed towards mining useful information from a large amount of unformatted data present in online blogs and online forums. We discuss an implementation of such a blog mining application. The application is broadly divided into two parts, the indexing process and the search module. Blogs pertaining to different organizations are fetched from a particular blog domain on the Internet. After analyzing the textual content of these blogs they are assigned a sentiment rating. Specific data from such blogs along with their sentiment ratings are then indexed on the physical hard drive. The search module searches through these indexes at run time for the input organization name and produces a list of blogs conveying both positive and negative sentiments about the organization.

1 Introduction

A blog, which is a collection of web pages on some website or portal, serves as an online diary maintained by an individual to share his thoughts and ideas and express his feelings. Blogs are powerful in the sense that they allow individuals over the globe to bring forth their ideas and garner feedback from other Internet users. Blogs appeared in the late 1990s but have since seen an unprecedented increase. Given the astounding blog-posting frequency and the amount of information communicated through online blogs, they are being viewed as potentially valuable resources of research. Furthermore, many people seem to get their news and form their opinions from authoritative blogs instead of standard media outlets, like broadcast news and newspapers [3].

Blog Mining techniques serve as an effective tool in social network analysis, economic research and network theory and form the basis for a myriad of services offered by popular blog analysis engines. For example, as a step towards social community mining, blog mining techniques can be used to find a community of bloggers who share a similar topic distribution in their blogs. This concept

P. Perner (Ed.): MLDM 2007, LNAI 4571, pp. 379–391, 2007.

is referred to as latent friend mining, wherein a latent friend is one who shares similar interests [2]. Determining communities of web pages based on named entity terms is another area where blog mining can prove to be useful. Named entity terms are names of persons, organizations, locations, etc. that occur in web documents with some relationships between them. Named entity terms are of high interest in web and blog search. While query strings can vary from a product model to a scientific concept, named entity terms are among the most frequently searched terms on the web [5].

"Reputation Management," is another technique under development by several companies [1]. Companies have now become aware of the potential of blogs to hurt them and have taken a two-pronged approach to dealing with the problem. On the one hand, many companies have created their own corporate blogs which have multiple purposes of keeping their customers and users informed, involved, and in some cases to garner feedback about products before releasing them to a wider audience. On the other hand, new blog mining startups have been created, which are hired by major companies to keep an eye on the blog space [8].

This paper discusses the latter kind of sentiment mining of the blog text. The architecture of the system is detailed in Figure 1.

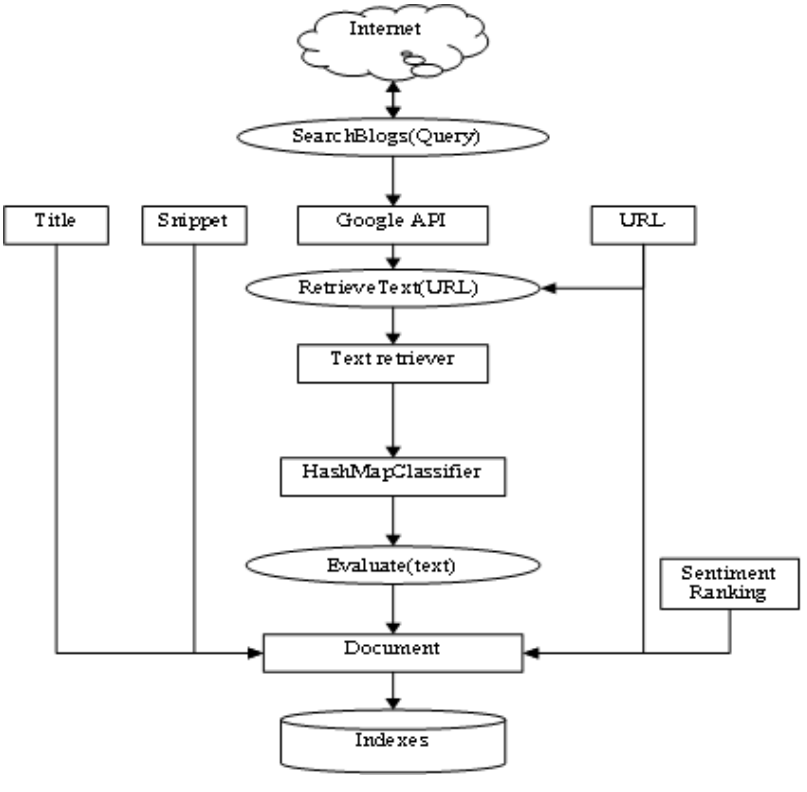

Fig. 1. System Architecture

The strength of positive or negative feelings is expressed by the sentiment ranking number that our program assigns to each blog. This sentiment ranking is a logarithmic number of positive or negative word counts from the blog text. The whole procedure of classifying a blog and giving it a ranking is done at index time, i.e. blogs are ranked and indexed prior to search time. This technique reduces the time required for searching results as they are already stored on the machine.

2 Blog Search

To search for blogs of a specific organization on the Internet we use APIs provided by Google. Google API is a lightweight framework developed for searching, made available by Google [6]. For implementation purposes, the Internet blog domain used is blogspot.com. The format of the query supplied to Google APIs is "company-name site:blogspot.com". The company names are obtained from a database table storing the list of Fortune 500 companies. The results returned are sorted according to Google's proprietary Page Rank method.

Each result returned consists of several parts as shown in Figure 2. These parts are:

summary - If the search result has a listing in the Open Directory Project directory, the ODP summary appears here as a text string. The Open Directory Project is a web directory of Internet resources. A web directory is something akin to a huge reference library. This directory is hierarchically arranged by subject - from broad to specific. The ODP is maintained by community editors who evaluate sites for inclusion in the directory. In Figure 2, the result does not have a summary from ODP.

URL - The URL of the search result, returned as text, with an absolute URL path. In Figure 2, "planetmath.org/encyclopedia/TimeComplexity.html" is a URL to reach this page.

snippet - A text excerpt from the results page that shows the query in context as it appears on the matching results page. This is formatted HTML and query terms are highlighted in bold in the results, and line breaks are included for proper text wrapping. If Google searched for stemmed variants of the query terms using its proprietary technology, those terms are also highlighted in bold in the snippet. Note that the query term does not always appear in the snippet. In Figure 2, "Time complexity refers to a function describing how much time it will take an algorithm ... The exact expression for time complexity of a particular sorting..." is the snippet. Snippet and summary are used interchangeably.

title - The title of the search result, returned as HTML. "PlanetMath: time complexity" is the title of the result shown in Figure 2.

cachedSize - Text (Integer + "k"). Indicates that a cached version of the URL is available; size is indicated in kilobytes. In the figure 22k is the size of the cached page.

relatedInformationPresent - Boolean indicating that the "related:" query term is supported for this URL. This is not a visible component of the results returned by Google.

hostName - When filtering occurs, a maximum of two results from any given host are returned. When this occurs, the second resultElement that comes from that host contains the host name in this parameter.

directoryTitle - If the URL for this resultElement is contained in the ODP directory, the title that appears in the directory appears here as a text string. Note that the directoryTitle may be different from the URL. [6]

PlanetMath: **time complexity**
Time complexity refers to a function describing how much **time** it will take an algorithm **...** The exact expression for **time complexity** of a particular sorting **...**
planetmath.org/encyclopedia/TimeComplexity.html - 22k - Cached - Similar pages

Fig. 2. Format of a result returned by Google API

3 Relevant Text Extraction

The URL of each result is used, in our program, to extract the text content of the corresponding web page. For this purpose we have used JTidy. JTidy is a Java port of HTML Tidy, an HTML syntax checker and pretty printer. Like its non-Java cousin, JTidy can be used as a tool for cleaning up malformed and faulty HTML. In addition, JTidy provides a Document Object Model interface (DOM) to the document that is being processed, which effectively makes you able to use JTidy as a DOM parser for real-world HTML [11]. The text retriever module uses JTidy to get a DOM representation of the web page and then extract text from it.

As seen on many blog portals, it is quite common to have several blogs listed on a single web page. These blogs are mostly written on different topics and are not related to each other. Of these, only a single topic or a few more may be relevant to the organization we are interested in. Doing a classification on all words of the web page yields highly erroneous results.

Hence, after fetching the text of the entire blog page, it is further processed to fetch pieces of text relevant to the corresponding organization. A record of the association between the query i.e. organization name and links corresponding to it, maintained during the Blog Search procedure described in Section 2, is utilized to extract such relevant text blocks. The complete text is split into various blocks with the company name as the delimiter. If the company name consists of more than one word then each individual block obtained in the first iteration is split further using any of the individual words in the company name as a delimiter.

For example, consider the text of a web page fetched for “Morgan Chase.” The process of extracting relevant text is clarified in Figure 3.

After dividing the text into different blocks, the process of retrieving relevant text starts. Since there is a low probability that a ”large” piece of text located before the first occurrence of the organization name would actually describe the organization, only 30 words are considered from the first block. Thereafter, each block is examined to check if its length is greater than 150 words. If no, the

entire block is considered relevant. If yes, only the first 75 words from both the start and end of the block are considered relevant. Finally, for the last block, the first 75 words from the beginning of the block are taken. If the size of the last block is less than 75 words the entire block is used. To check for organization name matches within the text of the web page, regular expressions have been utilized which further improves the accuracy of the text extraction process.

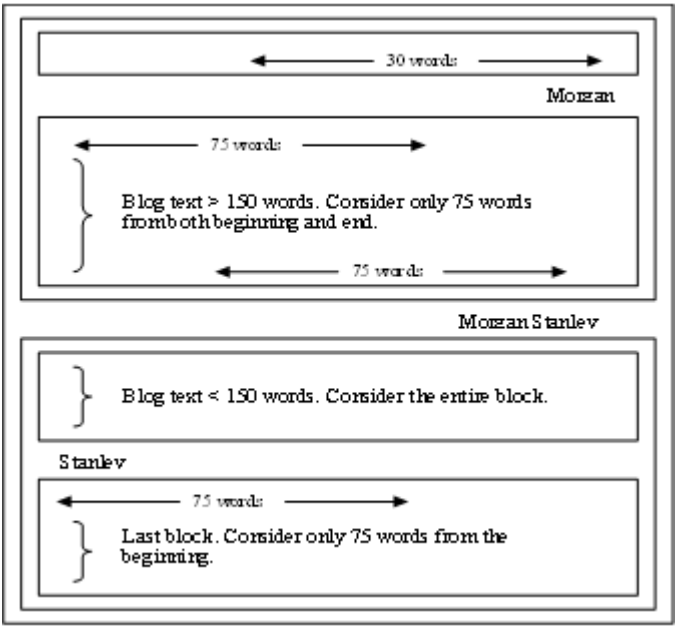

Fig. 3. Algorithm for extracting the relevant pieces of text

A sample run of the RelevantTextExtractor module on the content of a particular piece of blog text extracted from the web is shown in Figures 4 and 5. As shown in Figure 4, the blog is titled "J.P. Morgan posts 68% rise in quarterly profit" and the relevant text being extracted pertains to the company JP Morgan Chase. The RelevatTextExtractor module uses the procedure described in Figure 3 to search for occurrences of the phrase "Morgan Chase" or each individual word "Morgan" or "Chase" within the text, divide the text into blocks based on these occurrences and extract pieces of relevant text from the blocks. The output of the RelevantTextExtractor module is shown in Figure 5.

4 HashMap Classifier

4.1 Implementation

In order to determine the sentiment ranking of a piece of blog text, a table of words conveying either positive or negative sentiments was implemented. The

J.P. Morgan posts 68% rise in quarterly profit

J.P. Morgan Chase & Co. said its quarterly earnings soared 68% on strong investment banking growth and a gain from the sale of the bank's corporate trust business. But credit quality weakened somewhat, as it has at other major banks, suggesting that both commercial and individual customers are having more trouble keeping up with their bills. The Manhattan-based bank, the nation's third largest, earned $4.53 billion, or $1.26 a share, in the fourth quarter, up from $2.7 billion, or 76 cents a share, a year earlier. Excluding the $622 million after-tax gain on the sale of its trust business, net income was $3.9 billion, or $1.09 a share. Revenue rose 19% to $16.05 billion.

Analysts had projected a profit of 95 cents a share. Some analysts believe the weakening housing market and slowing economy are affecting credit quality and worry it could worsen in coming months. Mr. Dimon has been warning for some time that the stellar credit conditions -- which reflected a growing economy and reform of national bankruptcy laws in the fall of 2005 -- could not continue. In fact, weaker credit quality has begun showing in several divisions.

J.P. Morgan's strongest fourth-quarter performance came from the investment bank, where net income grew 51% to $1 billion in the latest quarter. Still, the provision for credit losses in this division rose to $63 million from $7 million in the third quarter.

Fig. 4. Sample blog pertaining to the company J. P. Morgan Chase

Block 0---> J.P.

Block 1---> posts 68% rise in quarterly profit J.P.

Block 2---> & Co. said its quarterly earnings soared 68% on strong investment banking growth and a gain from the sale of the bank's corporate trust business. But credit quality weakened somewhat, as it has at other major banks, suggesting that both commercial and individual customers are having more trouble keeping up with their bills.The Manhattan-based bank, the nation's third largest, earned $4.53 billion, or $1.26 a share, in the fourth quarter, up from $2.7 billion,

Block 3---> a profit of 95 cents a share. Some analysts believe the weakening housing market and slowing economy are affecting credit quality and worry it could worsen in coming months. Mr. Dimon has been warning for some time that the stellar credit conditions -- which reflected a growing economy and reform of national bankruptcy laws in the fall of 2005 -- could not continue. In fact, weaker credit quality has begun showing in several divisions. J.P.

Block 4---> 's strongest fourth-quarter performance came from the investment bank, where net income grew 51% to $1 billion in the latest quarter. Still, the provision for credit losses in this division rose to $63 million from $7 million in the third quarter.

Fig. 5. Set of relevant text blocks extracted from Figure 4

table contains the actual word, its category i.e. positive or negative and its weight which is a numerical value ranging between -1.5 and +1.5 in increments of 0.25, excluding 0. Weights less than 0 are assigned to words that convey negative sentiments and those greater than 0 are given to words that convey positive sentiments. The value of the weight indicates the extent to which a word is positive or negative. The table, at present, stores over 400 positive and negative words. We initially settled on 6 positive and 6 negative steps. To avoid "infinite" decimal numbers we chose the limit of 1.5 instead of 1, which will always provide exact decimal numbers.

When starting the sentiment mining process on a given blog, a hash table is created in memory, which contains the words conveying sentiments as keys and weight numbers between -1.5 and 1.5 as values. The information loaded into this hash table is extracted from the table mentioned above, which is implemented in Oracle (see Figure 6). Every word in the relevant text blocks of a blog, extracted as explained in Section 3, is then checked to see if it matches with the keys in the hash map. Whenever a blog word matches a hash table key, the weight for this key is returned. The weight of the entire set of relevant text blocks is obtained by calculating the sum of weights of individual words that have a match in the hash map. A negative word weight lowers the blog weight and a positive word weight adds to it. Finally, the logarithm of the absolute value of the resultant sum is

used as the sentiment rating for the blog text. The complete blog is classified as positive or negative depending on whether the resultant logarithmic value is greater or less than zero, respectively. If the resultant value is 0 then the blog is neutral, in which case, it is skipped. However, this is a rare condition and the probability of occurrence of such blogs is minimal.

To get a closer match between the words in the text and those in the hash map, the base words and their variations have been included in the database. An example of this is also shown in the figure 6.

As shown in Figure 6, both "loves" and "loved" give a match with "love" in the hash map. If a particular word in the text is not found in the map, a check is performed to see if the word ends with any of the common suffixes. If a common suffix is found, it is stripped and the remaining part of the word is again checked for a match in the hash map. Common suffixes of 1, 2, 3 and 4 character lengths have been used for this purpose. These are listed in Figure 7. This helps limit the size of the wordlist table by reducing the number of variations to be entered for each word in the database. It also helps achieve better classification due to the probability of getting more matches.

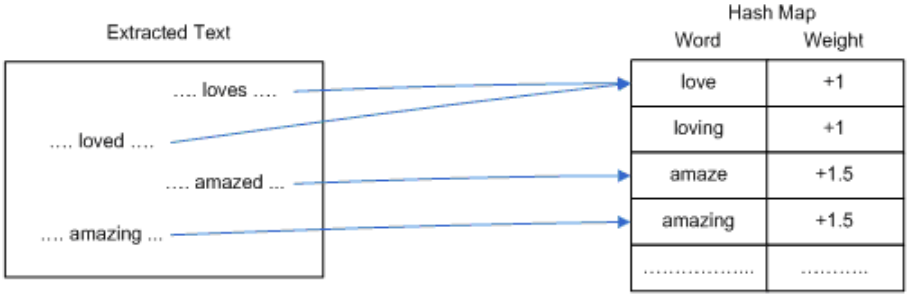

Fig. 6. Text word match with hash map keywords

4.2 Comparison of Classifiers

We have compared our HashMap classifier with two other classifiers, Dynamic Language Model Classifier and Binary Language Model Classifier, which are included in the LingPipe framework. LingPipe is a suite of natural language processing tools that performs tokenization, sentence detection, named entity detection, co-reference resolution, classification, clustering, part-of-speech tagging, general chunking, and fuzzy dictionary matching [12].

In order to compare our results with existing research, we needed a set of test documents which are publicly accessible and have already been classified as positive or negative. For that purpose, we have used LingPipe's language classification framework to perform the classification task on a set of test data, which is actually a collection of positive and negative movie reviews. Lee and

Suffixes having a single character length	s, d, y
Suffixes having a length of 2 characters	ed, er, ly
Suffixes having a length of 3 characters	ing, ent, ism, ive, ful, ous, ity
Suffixes having a length of 4 characters	ness, able, less, ment

Fig. 7. Table of common suffixes used

Pang have provided such movie review data for testing purposes. This data is already divided into two segments (slices), positive and negative. These reviews can be obtained from the link http://www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz. This test data set contains a total of 2000 movie reviews, 1000 with positive sentiments and 1000 with negative sentiments. Following standard machine learning methods, for the purpose of training, 1800 movie reviews, 900 positive and 900 negative, are used. For testing, the remaining 200 movie reviews are used.

For each classifier and each training case, the number of training characters, the time to train, the correctness of the tests, and the total time for testing are given in Figure 8. The five cases of the experiment were done with different training and test data sets. For each case, 1800 training files were randomly picked and both the classifiers were trained with them. The remaining 200 files were used to test those trained classifiers. The highest accuracy achieved with the Dynamic-LM Classifier was about 83%, while the lowest was 77%. On the other hand, the accuracy of the BinaryLMClassifier was always 100%. The Binary-LM Classifiers also takes less time for training because it gets trained on only one category. This means that if the DynamicLMClassifier was trained on 1800 files then the Binary-LM Classifier was trained only on 900 files.

The Dynamic Language Model Classifier (DLM) is slow in both cases of training and evaluation. But the advantage of the Dynamic classifier against the Binary LM Classifier (BLM) is that it can support n classifications. That means that DLM can be trained to classify sentiments into several categories such as positive, extremely positive, neutral, negative, etc.

On the other hand the BLM is very fast. As the name suggests, it can classify only two categories, positive and negative. BLM only gets trained on one category and it rejects the other category. When a file is fed to it, it returns "true" or "false," true if the file belongs to the category it was trained on. For every other category it returns false. So, if there are only two categories this model works perfectly fine with a high level of precision.

We used our HashMap classifier for the movie review data set, without adapting the key words to the movie domain. We only used generic sentiment terms as we would use for any kind of blog mining. Looking at the results in Figure 8, it becomes clear that the HashMap classifier does not require any training time, as opposed to the Dynamic-LM and the Binary-LM classifier. Its classification performance of 66% is below the results for both the classifiers. However, HashMap classifier runs faster than both classifiers even during testing.

	Training Cases	Training Chars	Training Time (ms)	Evaluation Cases	Correct Evaluations	% Correct	Time to evaluate (ms)
Dynamic LM-Classifier							
Case 1	1800	7037315	138282	200	154	77	29375
Case 2	1800	7039594	138609	200	156	78	26610
Case 3	1800	7038941	143578	200	157	78.5	28891
Case 4	1800	7045936	163297	200	165	82.5	29344
Case 5	1800	7000543	140828	200	158	79.5	29047
Binary LM-Classifier							
Case 1	1800	7037315	13563	200	200	100	2625
Case 2	1800	7039594	4125	200	200	100	1109
Case 3	1800	7038941	4469	200	200	100	1218
Case 4	1800	7045936	4109	200	200	100	1250
Case 5	1800	7000543	4094	200	200	100	1297
HashMap Classifier	0	0	0	2000	1320	66	7140

Fig. 8. Statistics for different classifiers

Note that in Figure 8 the time for our HashMap classifier is given for 2000 movie reviews, while the times for the two other classifiers are given for 200 only. Thus, if we want to compare equal work loads, we need to divide the run time of 7140 ms by 10, giving 714 ms, which is faster than the fastest time of the Binary-LM Classifier.

5 Lucene Indexing

5.1 Indexing Records

The next step consists of indexing the Google API results with the sentiment rank. In order to do the indexing we have used Lucene [10]. Lucene is a free, open source information retrieval API originally implemented in Java by Doug Cutting. It is supported by the Apache Software Foundation and is released under the Apache Software License. While suitable for any application which requires full text indexing and searching capability, Lucene has been widely recognized for its utility in the implementation of Internet search engines and local, single-site searching.

At the core of Lucene is an index. Although Lucene is used for text indexing it does not index files, it indexes document objects. A document is a collection of fields which are nothing but name value pairs. An index in turn contains a set of documents.

The following piece of code demonstrates how document objects have been created in our module.

```
public Document indexThis(IndexWriter writer, String query,
GoogleSearchResultElement r, HashMap wordWeightMap)
throws Exception
{
    Document doc = new Document();
    Double reverseBoost;

    RelevantTextExtractor rte = new RelevantTextExtractor();

    String url = r.getURL().replaceAll("<b>", "");
    String snippet = r.getSnippet().replaceAll("<b>", "");
    String title = r.getTitle().replaceAll("</b>", "");

    String text = rte.parseThis(query, url).toString();
    Double boost =
    new Double(HashMapClassifier.evaluate(text, wordWeightMap));

    if(boost.doubleValue() >= 0)
        reverseBoost = new Double(100.0 - boost.doubleValue());
    else
        reverseBoost = new Double(100.0 + boost.doubleValue());

    doc.add(Field.Text("url", url));
    doc.add(Field.Text("snippet", snippet));
    doc.add(Field.Text("title", title));
    doc.add(Field.Text("rating", boost.toString()));
    doc.add(Field.Text("reverse_rating",
                        reverseBoost.toString()));

    return doc;
}
```

As shown in the code segment, a new instance of document object is created. The URL, snippet and title are extracted from each search result returned by the Google API. The URL is then passed to the RelevantTextExtractor module to retrieve the entire text of the blog web page and extract relevant text content from it. This relevant text is then passed to a static "evaluate" function of the HashMapClassifier class which returns a sentiment rating of the blog text. A reverse sentiment rating is also calculated, which is useful to sort blogs in descending order of their sentiment ratings. A document object is then created using the values of URL, snippet, title, rating and reverse rating. The document is then added to a new index or appended to one if the index already exists.

5.2 Searching Indexes

The search module provides a simple JSP page allowing a user to enter an organization name and search for it in the records already indexed on the hard drive. The snippet fields of all documents are checked for a match with the organization name. Once such records are found, positive and negative blogs

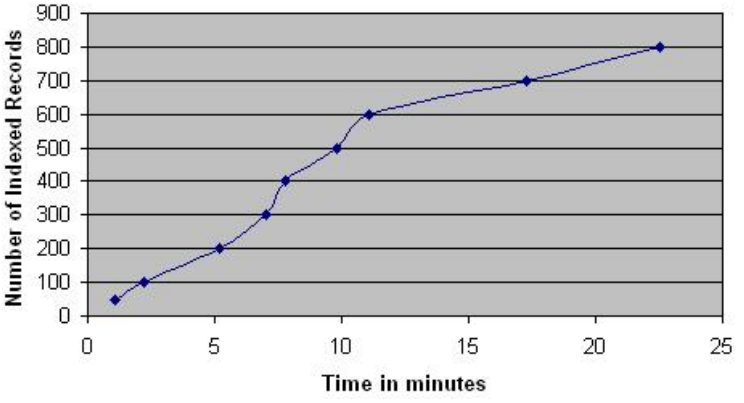

Fig. 9. Indexing Speed

are sorted and displayed separately on the JSP page. The displayed records are again in the form of small documents with a title, sentiment rating, snippet and URL.

6 Performance Statistics

Figure 9 shows the time required to complete the entire indexing process. This includes fetching blog URLs using Google APIs, extracting text from the blog web pages, determining sentiment rating to classify them as well as indexing blog data. To index blogs from about 800 web pages takes just over 22 minutes, which is fast considering the amount of processing that is done before indexing. Moreover, this does not affect run time performance since indexing is done offline, before hand.

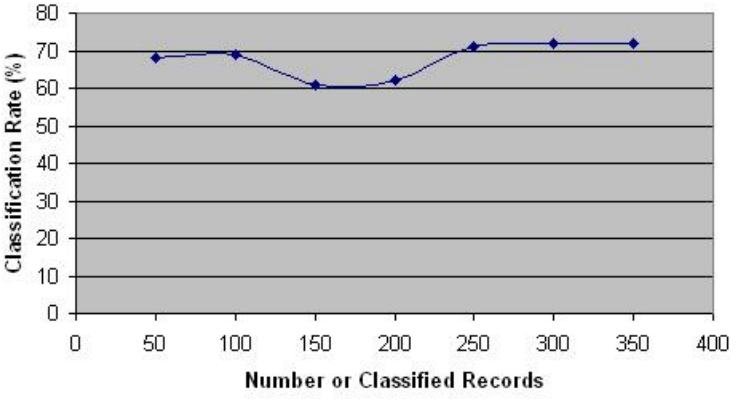

Fig. 10. Classification Rate

Figure 10 gives a plot of the number of blogs classified versus the percentage of correct classifications. As seen from the figure, this varies between 60-73%. Also, as seen from the comparison table in Figure 8, the average classification rate for 2000 movie reviews is 66%.

7 Conclusions and Future Work

For companies trying to determine where they stand in the market with respect to their customers, suppliers and many other stakeholders, blogs are becoming a prime medium. With the help of blog mining software, employees can easily go through the existing blogs to gain an insight into the feelings and opinions of users about the organization. This helps them make important decisions on improving product quality, increasing profit, market standing, and customer satisfaction.

Our implementation performs this kind of sentiment mining on blogs. The URLs of blogs specific to an organization are first fetched using APIs provided by Google. The entire text of the blog page corresponding to each URL is then extracted. The size of text is further reduced by determining pieces of text relevant to the organization. Using a table of words conveying sentiments the sentiment rating of the blog is determined and the blog is categorized as either positive or negative. The data for each blog is then indexed using Lucene APIs and stored on the hard drive. Since this entire classification takes place offline, the run time search procedure only needs to search through the indexed records already created, thereby keeping access time to a minimum. In our implementation, we have currently indexed 100 blogs for each of the Fortune 500 companies from the domain blogspot.com.

Using a hash map data structure for classification of blogs makes the indexing process very fast. However, since the classification relies only upon a list of positive and negative words the accuracy achieved is about 60-75%. To improve accuracy the size of the word list in the database needs to be increased by adding new positive and negative words as and when found. Note that due to the efficient retrieval of data from hash tables, adding new terms will, on an average, have a small effect on evaluation time. Thus, an expected increase in classification accuracy will maintain the positive timing characteristics of our approach. As an additional enhancement, a table of the most commonly occurring phrases conveying sentiments along with their weights can be created. An initial text weight can be determined first by checking for these common phrases in the extracted relevant text. These phrases can then be eliminated from the text and remaining text can be classified using our Hash Map classifier. This would further refine the classification task. Moreover, in addition to searching for only the company name, its products and/or services can also be used to fetch relevant text. This would result in a larger text on which classification would be more accurate.

References

1. Aschenbrenner, A., Miksch, S.: Blog Mining in a Corporate Environment, Technical Report ASGAARD-TR-2005-11, Technical University Vienna (September 2005) (accessed February 1, 2007), http://ieg.ifs.tuwien.ac.at/techreports/Asgaard-TR-2005-11.pdf
2. Shen, D., Sun, J.-T., Yang, Q., Chen, Z.: Latent Friend Mining from Blog Data. In: International Conference on Data Mining, pp. 552–561 (2006)
3. Tirapat, T., Espiritu, C., Stroulia, E.: Taking the community's pulse: one blog at a time. In: International Conference on Web Engineering, pp. 169–176 (2006)
4. Mishne, G.: Experiments with Mood Classification in Blog Posts. In: Style2005 the 1st Workshop on Stylistic Analysis of Text for Information Access, at SIGIR 2005 (August 2005)
5. Li, X., Liu, B., Yu, P.S.: Mining Community Structure of Named Entities from Web Pages and Blogs. In: AAAI Spring Symposium, Computational Approaches to Analyzing Weblogs, pp. 108–114 (2006)
6. Google Web APIs (March 10, 2006), http://code.google.com/apis
7. Fischer, I., Torres, E.: A Distributed Blog Search Platform (2006)
8. The Blog in the Corporate Machine, The Economist, (February 11, 2006)
9. Fortune 500 Full List, CNNMoney (April 17, 2006), http://money.cnn.com/magazines/fortune/fortune500/full_list
10. Hatcher, E., Gospodnetic, O.: Lucene in Action (2006)
11. JTidy - HTML Parser and Pretty-Printer in Java (March 10, 2006), http://jtidy.sourceforge.net
12. LingPipe (2007), http://www.alias-i.com/lingpipe