J. Indian Inst. Sci., Sept.–Oct. 2006, **86**, 481–492

Intelligent web monitoring–A hypertext mining-based approach

MANAS A. PATHAK* AND VIVEK S. THAKRE
Department of Computer Science & Engineering, Visvesvaraya National Institute of Technology, Nagpur 440 010, India.
email: manas07@gmail.com; Phone: 0712-2533516.

Received on August 30, 2006.

Abstract

The World Wide Web has become one of the principal sources of information since its inception. With large amount of content added and deleted, the amount of change in hypertextual data is massive. This rapidly changing nature of the WWW makes the task of tracking information intractable when done manually. In this paper we propose an approach for intelligently monitoring the website for changes, taking into consideration the user interests and ranking of these changes according to relevance. A prototype system WebMon based on this approach is presented.

WebMon consists of basic components performing infrastructural activities such as crawlers and indexers. Also it takes as input keyword weights based on the user interests. It then represents the hypertextual data in the website in the form of a vector space model (VSM). Periodically this process is carried out to get the VSM representing the hypertextual data of the website at that instance of time. To monitor for changes, the data in VSMs at different instances of time is compared and the corresponding changes are ranked according to their relevance according to the user. A modified nearest neighbor algorithm (NN) is implemented for the same. To further improve the accuracy and self-adjustability of the relevance rankings, the system employs a modified supervised learning algorithm thereby taking into account the behavior of the user intelligently.

The WebMon system has been tested extensively on many websites giving results as expected. In this paper we report some experimental results showing the effectiveness of the proposed approach.

Keywords: Relevance ranking, vector space model, nearest neighbors, supervised learning.

1. Introduction

Since its advent, the World Wide Web (WWW or W^3) has become one of the principal sources of information. Having over 350 million pages, it continues to grow rapidly at a million pages per day [1]. Apart from its growth rate, the amount of change in textual data is a massive 600 GB per every month [2]. Even if we consider a small subset of the web, as in a moderately sized website, the amount of information and the degree by which it changes is still by far beyond human comprehension.

In many cases, the need for searching and monitoring the information arises. Students may want to monitor a university website for admission updates, businessmen may want to monitor a bank website for news about interest rates, and hobbyists may want to monitor

*Author for correspondence.

their favorite blogs for the latest happenings. As mentioned earlier, it is almost impossible to do this manually for a moderately sized website or even for a few small websites taken together.

However, it is not too difficult to build a system which just crawls through the web periodically and looks up the document meta-data for changes. If this approach would suffice, there would not be much need for it. Fortunately or unfortunately, this is not the case because if the changes are in large number, just a listing of the changes would not help much. Here we consider some techniques by which the listing of changes would be ordered according to the *relevance* of the user.

The approach assumes that the user provides the keywords on which the given website has to be monitored. In the following discussion, we normally discuss about monitoring only one website, but it should be kept in mind that this approach can be easily modified to monitor multiple websites at once. This paper explores the above thesis by proposing and evaluating several metrics and algorithms relevant to the monitoring task and presents the prototype WebMon system. Also a modified supervised learning algorithm is developed to give more accurate and self-adjusting relevance rankings.

2. Problem formulation

As summarized in the previous section, the WebMon system provides experimental support to our thesis that a system can automatically monitor the web and then order the changes according to the user relevance. Later sections describe in detail as to how this task is carried out. In this section we consider the precise formulation and representational assumptions that underlie our approach.

Given:
1. Starting URL(s) of the website(s) to be monitored
2. User-defined keywords and their interests
3. User behavior with respect to the listed changes.

Determine:
A relevance ranking of the changes occurred in the website periodically.

3. Infrastructural activities

In WebMon some infrastructural activities are necessary to convert hypertextual data into a representational model from which analysis can be performed and useful information can be extracted. These are briefly described below. These activities can be carried out one after another or simultaneously. In the latter case, some additional care has to be taken to ensure proper synchronization.

3.1. *Crawling*

In a website, the data is present in the form of documents which are connected together using *hyperlinks*. In every document, there are hyperlinks which link to other documents. However, there is no systematic catalog of documents in which all of them would be listed.

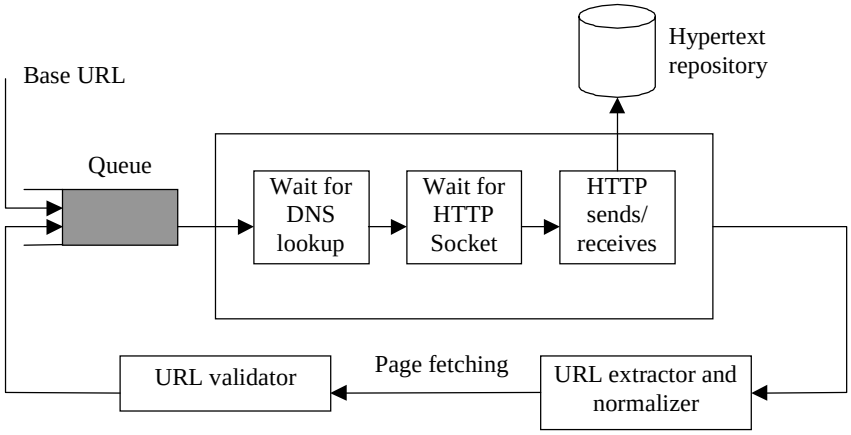

FIG. 1. Functional architecture of a crawler.

Hence to get all the data an activity called *crawling* is performed using a software agent called *crawler*.

The entire life cycle of the crawler is explained in Fig. 1. The crawler takes the base URL as input and inserts it in the queue. After this the crawler deletes one URL from the queue. In the page-fetching component, a DNS lookup is made to get the address of the corresponding document. An HTTP socket is established and then the request for the document content is made. Once the document content is obtained, it is stored in the hypertext repository which is made available to other modules (see indexer). Also the document content is given to the URL extractor which parses it for outgoing links and then normalizes these URLs (e.g. converts relative links to absolute links). After this each of the newly found URLs is given to the URL validator which checks to see whether the URL is already crawled and if it belongs to the same domain. All the valid URLs are added back to the queue. This process is continued till the queue is empty.

It is seen that a major bottleneck, which negatively affects the efficiency of WebMon, is crawling. For the system to be scalable, it is necessary to crawl rapidly. To meet this end, techniques like multithreading, connection pooling, DNS cache are implemented [3, 4]. Also some commercial crawlers which can crawl more than 100 documents per second are available [5]. There is also a class of crawlers which selectively seeks out pages relevant to a predefined set of topics called focused crawlers [6]. These crawlers give much better performance as they first analyze the crawl boundary to find the links that are likely to be most relevant for the crawl, thus avoiding irrelevant regions of the web.

3.2. *Indexing*

The indexing of the website is done by another software agent called indexer (Fig. 2). After crawling, the contents of documents are obtained from the repository one by one.

The document content is given as input to the filter component which first splits the text into individual words and eliminates punctuation marks and other unnecessary symbols. It

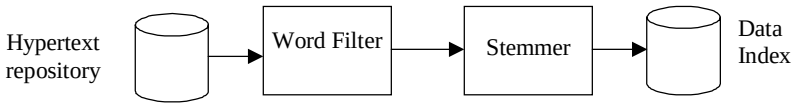

FIG. 2. Block diagram of indexing process.

then proceeds to remove the stop words appearing in the text. This is done by using standard set of noise word vocabularies [7]. An optional word stemmer component is then used to stem each word to its root (e.g. running is stemmed to run). Common stemming methods use a combination of morphological analysis (e.g. Porter's algorithm [8]) and dictionary lookup (e.g. WordNet [9]). Other corpus-based analysis of word variants can be used to enhance the performance of stemming algorithms [10].

After this analysis, we store the meta-data about each keyword (i.e. document where it is located, and its offset from the start of the document) into a *data index*. Even if a keyword occurs more than once in a document, it is still stored twice in the data index and the reasons for it will be clear later on. An effective data index structure is vital for high performance information retrieval systems. Documents that contain a specified keyword can be efficiently located by using an *inverted index* [4], which maps each keyword k to the set S of the documents that contain k. Since such indices must be stored on disk, the structure should attempt to minimize the number of I/O operations [11]. Various optimized data index organizations are used to meet this end [12, 13]. In very large-scale indexes, file systems are widely used. One such prominent implementation is the Google File System, which provides a scalable distributed file system for large distributed data-intensive applications [14].

From the HTML content, the indexer can optionally proceed to create a partial *document tag tree* [15]. This structure contains the textual content and the tag with which it is associated. This is so because the text contained in these tags will have a relatively higher importance as compared with other text.

4. Keyword weights and user interests

For intelligent web monitoring, all of the keywords are not treated equally as some are more relevant than others. To achieve this we assign *weights* to individual keywords. Again it should be kept in mind that relevance ranking is not an exact science, but these are some well-accepted approaches [11].

In this paper, we consider three types of weights assigned to a keyword. They are as follows:

1. Raw statistical weight (W_R)
2. User-defined weight (W_U)
3. User feedback weight (W_F)

The total weight can be empirically calculated as:

$$W = a_R W_R + a_U W_U + a_F W_F$$

where a_R, a_U and a_F are the weight coefficients having values between 0 and 1. We can set their values to fine-tune the keyword's relative importance.

For instance, if the user is more interested in the changes one specifies, then a higher value of a_U (say $a_U = 0.85$, $a_R = 0.4$) can be chosen. On the other hand, if the user is more open to other changes, then a comparatively lower value a_U (say $a_U = 0.5$, $a_R = 0.8$) can be chosen.

As stated earlier, the user is given the facility to directly specify the keywords of interest on which monitoring has to be done. Some nonzero user-defined weights are assigned to these keywords according to their priority given by the user. These weights directly specify the user's interests and are given higher share in the overall weight of a keyword. For the rest of the keywords, this value is zero. The user feedback weights are considered later (see Section 8). For now, it can be assumed that its value is initially zero for all keywords.

The weights W_U and W_F are user-dependent weights and are evaluated for each user for each website. The weight W_R is user independent and is calculated once for every website.

4.1 *Calculation of raw statistical weight and relevance*

To calculate the keyword relevance, the basic question to be answered is how relevant is a document with respect to a given keyword. Simply counting the number of occurrences of the keyword in the document is usually not a good indicator as it heavily depends on the length of the document and the relevance does not increase linearly with the frequency of occurrence. Hence, some other metrics are considered depending on the keyword occurrence as mentioned below.

Firstly, we discuss some parameters, which the calculation of the relevance metrics can be based on. These parameters can easily be calculated from the data index. For the rest of the discussion on keyword relevance, we use the symbols *k* and *d* to represent keywords and documents, respectively.

n(d) = number of keywords in document *d*
n(d, k) = number of occurrences of keyword k in document *d*
n(k) = number of documents in which the keyword *k* occurs at least once

The raw statistical weight of the keyword W_R is defined as the *inverse document frequency* (IDF), which is a measure of the general importance of the keyword. It is calculated as:

$$W_R = \frac{1}{n(k)}.$$

We also define a relevance metric *r* as follows.

$$r(d,k) = \log\left(1 + \frac{n(d,k)}{n(d)}\right).$$

Observe that this metric takes the length of the document into account as it depends on number of keywords $n(d)$. The relevance still grows with increasing values of $n(d, k)$; however, the increase is logarithmic. The log factor is used to avoid excessive weight to frequent keywords [11]. Also if the keyword *k* doesn't occur at all in document *d*, the relevance is log (1) = 0.

Many systems refine the above metric by using other information like the position of the keyword in the document. Also, the tag in which the keyword occurs can be considered, with the keywords occurring in tags like <title>, <h1>, etc. being given more importance as compared to other keywords. These notions can be formalized by some modifications to the above-mentioned formulae. In the information retrieval (IR) community, regardless of the exact formula, the relevance of the document with respect to a keyword is called as *term frequency* (TF) [11].

A weighted relevance function F is calculated as follows.

$$F(d,k) = r(d,k) \times W$$

where W is the total weight of the keyword k calculated as mentioned above.

The metric F gives a measure of the relevance of the keyword k with respect to the document d. Its effective value for any keyword will be scaled according to its weight. A more important keyword will have a higher value than a less important one.

5. Vector space model

The vector space model (VSM) is a mathematical model widely used for information filtering and information retrieval. It was used for the first time by the SMART Information Retrieval System [16]. It represents natural language documents in a formal manner by the use of vectors in a multi-dimensional space.

In VSM, we consider a multidimensional Euclidean space with each axis corresponding to a keyword. A document is conceptually represented by a vector of keywords extracted from the document, with associated weights representing the importance of the keywords in the document and within the whole document collection. The component of the document vector in a given keyword direction can be determined in many ways [17]. A common approach uses the so-called *tf – idf* method. Here we use the relevance metric F (discussed above) as it gives the keyword relevance scaled according to the keyword weights. In this way, if there are n keywords occurring in the website, we represent every document as vectors in the n-dimensional vector space to get a website matrix.

For the keywords $k_1, k_2, \ldots k_n$ and the document d, we have,

$$\vec{r} = F(d,k_1)\hat{k}_1 + F(d,k_2)\hat{k}_2 + \ldots + F(d,k_n)\hat{k}_n$$

where $\hat{k}_1, \hat{k}_2, \ldots \hat{k}_n$ are unit vectors in the respective keyword directions and $\vec{r}$ is the vector representing the document d. It should be understood that most of the components of these vectors will be zero. It is assumed that the keywords which we are considering are after elimination of stop words and other tasks.

6. Nearest neighbor algorithm (NN)

As mentioned above, the vector space model has been used extensively for information retrieval tasks. One of the basic operations to be performed is to find a document which is most similar to the given one. In the language of vectors, we have to find the nearest vector to the given vector in the same vector space. This is called as the *nearest neighbor algo-*

rithm (NN) [18]. It is primarily used in search engines, where the query string is taken as a document itself and the NNs to it are found out and ranked according to their 'nearness' (i.e. the geometric closeness to the given vector in the model). In this paper we do not consider query strings but use the NN approach to find the documents similar to a given document and the degree of difference between the two.

In VSM, we consider the angle between the two vectors as a measure of nearness between them. There exist alternative approaches to nearness, like geometric distance, city-block distance, etc. The advantage of this approach is that the relative sizes of the vectors are not considered. For example, even if a document is very long and another one is small they can still be quite near if their information content is the same.

Using the dot product, the angle between the two vectors $\vec{r}_1$ and $\vec{r}_2$ is calculated as:

$$\cos\theta = \frac{\vec{r}_1.\vec{r}_2}{|\vec{r}_1||\vec{r}_2|}.$$

7. Monitoring of a website

Using the above-mentioned concepts, we now proceed to the actual problem of monitoring the website. When the site is to be monitored, the website is represented as a matrix M in the vector space as mentioned above. Then periodically the same process is carried out and again a matrix M' is obtained. This matrix represents the new state of the information content of the website. To monitor the changes of the website, our job is to determine the changes in information content that may have occurred in the instances M and M'.

In this paper, we consider three types of changes which we are interested in monitoring. Here type 1 change is purely informational, while changes of type 2 and 3 are structural as well as informational.

Type 1: Addition and deletion of information content in a document
Type 2: Introduction of new documents in the new instance
Type 3: Disappearance of documents from the new instance.

7.1. *Type 1 changes*

First we will consider the changes of type 1. It can be assumed that in most websites, the structure does not change very frequently as compared to the information content. That is normally the information inside documents changes much more often than documents being added and removed from the website. Hence, the change of type 1 will be the most common of all the changes.

For monitoring type 1 changes, we first have to determine the documents which have undergone such a change. This is done by determining the documents in the original instance and the new instance. In vector parlance, we have to check for all those vectors which exist both in the original and in the new matrices. Here we note that we are currently not concerned about the components of the vectors but the URLs corresponding to them. In other words, given a vector $\vec{r}$ in M, for it to be a candidate for type 1 change, there should exist another vector $\vec{r}'$ in M' and the URL of the documents represented by both vectors should be the same.

Now using the NN approach we consider the similarity between the two vectors.

$$\cos\theta = \frac{\vec{r}_1.\vec{r}_2'}{|\vec{r}||\vec{r}'|}.$$

The measure cos θ gives the degree of similarity between the two vectors (and their corresponding documents). If the value of θ is small, then the change between the two documents or rather the two instances of the documents is also small and vice versa. It should be noted that this change reflects the change in the information content as the value θ is based on the concepts of relevance ranking, NN, etc. as mentioned earlier. Now if we rank the documents in the descending order of θ, we get the changes occurring in the website in the descending order of relevance.

In the output, along with the listing of the URLs where the changes have occurred, the keywords added/deleted are also mentioned. This is easily done as the data index contains all the keywords of both the earlier and new versions of the document.

7.2. *Type 2 and 3 changes*

The approach for monitoring type 2 and type 3 changes is similar. Firstly to determine such documents, we have to use an approach similar to the one used for type 1 changes. To determine the documents undergoing type 2 change, we have to get all the vectors which occur in M' but the vectors having the corresponding documents with the same URL do not occur in M. Similarly, for documents undergoing type 3 changes, we have to get all the vectors which occur in M but the vectors having the corresponding documents with the same URL do not occur in M'. To rank these documents, we simply consider the magnitude of the vectors corresponding to the documents. As the components of the vector are in fact of relevance to the corresponding document with respect to that keyword, empirically, the magnitude of the vector gives the total relevant information content of that document. Hence we list the documents in the descending order of the magnitude of their vectors. A document corresponding to a large magnitude value will symbolize a large information change and vice versa.

8. Learning from user's behavior

Till now, it was assumed that the changes in the website will be displayed and ranked according to relevance. This output was presented to the user who was allowed to view a change. Now we consider by learning from the user's behavior the system can accordingly improve the relevance of the output.

To improve the output, we use the concept of *supervised learning*. Supervised learning is a machine-learning technique for creating a function from training data. The training data consists of pairs of input objects, and desired outputs. The task of the supervised learner is to predict the value of the function for any valid input object after having seen a number of training examples (i.e. pairs of input and target output). In this case, input object is the entry selected from the listing by the user. From the selected entry we can assume that the user is interested in this 'change'. Therefore, the desired output is again a listing of changes

in which the ranking of that particular change is improved relatively. Here it should be noted that we are interested in the changes but not the documents in which the changes have occurred. If the same changes occur in another document, it would be ranked relatively higher the next time. Again, we consider changes in the document as the changes in the keywords of that document. For improving the relative ranking of that change, we introduce the concept of *user feedback weight*. If the user is interested in that change, we give some incentive to the keywords due to which that change has occurred.

The keywords present in the change between the two documents (which are selected by the user) should get some incentive weight as the user was interested in them. This incentive weight is proportionally distributed among the present keywords. A keyword present in the change will have a share in the incentive as the ratio of component of the vector in that keyword dimension to the magnitude of the sum of the component vectors in the direction of their respective keywords which are responsible for the change. This is the positive user feedback weight W_{F+}.

Let K_P be the set of keywords changed in the selected document. For each keyword k occurring in the selected document, we calculate:

$$W_{F+} = \frac{\vec{r} \cdot \hat{k}}{\sqrt{\sum_{\forall k_i \in K_P} (\vec{r} \cdot \hat{k}_i)^2}}$$

where $\vec{r}$ is the corresponding document vector, and $\hat{k}$, the unit vector in the direction corresponding to the keyword k.

Similarly, if the user selects the n^{th} entry, the system mistakenly anticipates that the user would have been interested in the first $(n-1)$ changes. To correct this, some disincentive weight is given to all the keywords present in each entry ranked higher than the chosen entry but do not occur in the selected change. The last condition is necessary as a selected keyword will be given a negative feedback multiple times despite it occurring in the change, thereby reducing its effective weight. This negative feedback weight W_{F-} is calculated similar to W_{F+}, only that it is negative in magnitude.

Let K_P be as defined above and K_Q be the set of keywords changed in the given document. For each document ranked above the selected document and for each keyword k in that document, we calculate:

$$W_{F-} = \begin{cases} 0 & \text{if } k \in K_P \\ -\dfrac{\vec{r} \cdot \hat{k}}{\sqrt{\sum_{\forall k_j \in K_Q} (\vec{r} \cdot \hat{k}_j)^2}}, & \text{otherwise} \end{cases}$$

where $\vec{r}$ is the corresponding document vector, and $\hat{k}$, the unit vector in the direction corresponding to the keyword k.

The user-feedback weight is then used for calculating the total weight of the keyword as mentioned earlier (see Section 4).

9. Case study

To provide an experimental verification of the above-mentioned theory, we now consider a sample website and monitor its changes using WebMon. A three-document website presented here is a scaled-down example of a typical financial information website which can be considered as an ideal application of WebMon. The user-defined weights of the keywords are shown in Table I. We assume the values for the weight coefficients a_R, a_U and a_F as 0.25, 0.6, and 0.85, respectively. Here only the textual content is shown here for clarity of illustration.

D1 = "An entity whose income exceeds its expenditure can lend or invest the excess income. On the other hand, an entity whose income is less than its expenditure can raise capital by borrowing or selling equity claims, decreasing its expenses, or increasing its income.

D2 = "Individual businesses are established in order to perform economic activities. With some exceptions (such as cooperatives, non-profit organizations and generally, institutions of government), businesses exist to produce profit. In other words, the owners and operators of a business have as one of their main objectives the receipt or generation of a financial returns in exchange for expending time, effort and capital."

D3 = "Today commerce involves a complex system of companies that try to maximize their profit by offering products and services to the market, at the lowest production cost. There is a system of world wide or foreign commerce, which some argue has gone too far."

9.1. *Vector space model*

After these documents are crawled and indexed, the vector space model is created. The vector space for the document *D*1 is as shown in Table II.

The vector spaces for other tables are similarly obtained. After a certain period of time, we again retrieve the above documents along with the modifications in them.

D1' = "An entity whose income exceeds its expenditure can lend or invest the excess income. On the other hand, an entity whose income is less than its expenditure can raise capital by borrowing or selling equity claims, decreasing its expenses, or increasing its income. The lender can find a borrower, a financial intermediary, such as a bank or buy notes or bonds in the bond market."

D2' = "Individual businesses are established in order to perform economic activities. With some exceptions (such as cooperatives, non-profit organizations and generally, institutions of government), businesses exist to produce profit."

D3' = "Today commerce involves a complex system of companies that try to maximize their profit by offering products and services to the market, which consists both of individuals and other companies, at the lowest production cost."

Table I
User-defined weights

Keyword	W_U
Profit	0.4
Bank	0.3
Business	0.2
Market	0.1

Table II
Vector space model

Keyword	$F(d, k)$	Keyword	$F(d, k)$
Borrowing	0.003545753	Hand	0.003545753
Capital	0.001969863	Income	0.164019635
Claims	0.003545753	Increasing	0.003545753
Decreasing	0.003545753	Invest	0.003545753
Entity	0.006952421	Lend	0.003545753
Equity	0.003545753	Less	0.003545753
Exceeds	0.003545753	Other	0.001772877
Excess	0.003545753	Raise	0.003545753
Expenditure	0.006952421	Selling	0.008864383
Expenses	0.003545753		

Table III
Relevance rankings in ascending order of cos θ

Document	Document	Cos θ
D1	D1′	0.539248207
D3	D3′	0.597266731
D2	D2′	0.75769057

Table IV
Feedback weights

Keyword	W_{F+}	Keyword	W_{F+}
Commerce	0.003367231	Offering	0.003367231
Companies	0.006584757	Other	0.001683616
Complex	0.003367231	Production	0.003367231
Consists	0.010101693	Products	0.003367231
Cost	0.003367231	Profits	0.003367231
Individuals	0.010101693	Services	0.003367231
Involves	0.003367231	System	0.003367231
Lowest	0.003367231	Today	0.003367231
Market	0.012753387	Try	0.003367231
Maximize	0.003367231	Which	0.003367231

9.3. *Relevance ranking*

We then use WebMon to determine the changes between the two website instances to get the following relevance ranking, which is in the decreasing order of cos θ.

9.4. *Learning*

Now in this relevance ranking, if the user selects the entry D3, then the positive feedback weights are added for the keywords as mentioned in Table IV.

For instance, in D3, there is a change due to keywords ‘individuals’, ‘consists’, etc. Hence, these keywords have been assigned higher feedback weights. In the next monitoring, these weights will be considered for the calculation of the relevance metrics. Similarly, the negative feedback weights are calculated for the keywords of D1.

10. Conclusion and future prospects of web monitoring

In this paper, we have demonstrated an approach for monitoring the changes of a website and ranking them according to the user interests. As far as web monitoring is concerned, the immediate goals are to improve the system efficiency and scalability. Another area which requires some research is to develop algorithms to decide which old documents should be re-crawled. That is based on the pattern of changes occurring in the website, the system should be able to predict beforehand the likelihood of the change occurring in the document.

Semantic web monitoring is another monitoring paradigm in which instead of taking keywords as input from the user, we take ontology. The ontology could be any structure which

represents the organization of the information represented in the website. Even if the changes do not occur directly in the *classes* in which the user is interested but their related classes, the changes will still be shown.

Acknowledgements

The authors would like to thank Prof. N. L. Sarda, Indian Institute of Technology, Mumbai for his valuable guidance.

References

1. K. Bharat, and A. Broder, A technique for measuring the relative size and overlap of public web search engines, *Proc. 7th World-Wide Web Conf.* (WWW7) (1998).
2. B. Kahle, Preserving the Internet, *Sci. Am.*, **276**, 82–83 (1997).
3. A. Heydon, and M. Najork, Mercator: A scalable, extensible web crawler, World Wide Web (1999).
4. S. Brin, and L, Page, *The anatomy of a large-scale hypertextual web search engine. WWW7/Computer Networks*, **30**, 107–117 (1998).
5. A. Heydon, and M. Najork, *Mercator: A scalable, extensible web crawler*, World Wide Web, **2**, 219–229 (1999).
6. S. Chakrabarti, M. van den Berg, and B. Dom, Focused crawling: A new approach to topic-specific web resource discovery, *8th WWW Conf.*, Toronto (1999).
7. http://dev.mysql.com/doc/refman/5.0/en/fulltext-stopwords.html
8. M. F. Porter, An algorithm for suffix stripping, *Program*, **14**, 130–137 (1980).
9. G. Miller, WordNet: a lexical database for English, *Commun. ACM*, **38**, 39–41 (1995).
10. J. Xu, and W. B. Croft, Corpus-based stemming using co-occurrence of word variants. *ACM Trans. Inf. Systems*, **16**, 61–81 (1998).
11. A. Silberchatz, H. Korth, and S. Sudarshan, *Database system concepts*, McGraw-Hill (1998).
12. S. Putz, *Using a relational database for an inverted text index*, Xerox Palo Alto Research Center, Technical Report SSL-91-20 (1991).
13. M. Lifantsev, and T. Chiueh, I/O-Conscious data preparation for large-scale web search engines. *Proc. 28th VLDB* (2002).
14. S. Ghemawat, H. Gobioff, and S. T. Leung, The Google file system, *ACM SIGOPS Operating Systems Rev.*, **37**(5), 29–43 (2003).
15. G. Pant, Deriving link-context from HTML tag tree, *8th ACM SIGMOD Workshop on Research Issues in Data mining and Knowledge Discovery* (DMKD 2003).
16. C. Buckley, *Implementation of the SMART information retrieval system*, TR85-686, Computer Science Department, Cornell University (1985).
17. D. L. Lee, H. Chuang, and K. Seamons, Document ranking and the vector-space model, *Software, IEEE*, **14**, 67–75 (1997).
18. N. Roussopoulos, S. Kelley, and F. Vincent, Nearest neighbor queries, *Proc. 1995 ACM SIGMOD Int. Conf., on Management of Data*, San Jose, CA (1995).