Automatic Extraction of Informative Blocks from Webpages

Sandip Debnath
Department of Computer Science
The Pennsylvania State University
University Park
PA 16802, USA
debnath@cse.psu.edu

Prasenjit Mitra
School of Information Sciences & Technology
The Pennsylvania State University
University Park
PA 16802, USA
pmitra@ist.psu.edu

C. Lee Giles
School of Information Sciences & Technology
The Pennsylvania State University
University Park
PA 16802, USA
giles@ist.psu.edu

ABSTRACT

Search engines crawl and index webpages depending upon their informative content. However, webpages contain items that cannot be classified as the "primary content", e.g., navigation sidebars, advertisements, copyright notices, etc. Most end-users search for the primary content, and largely do not seek the non-informative content. A tool that assists an end-user or application to search and process information from webpages automatically, must separate the "primary content blocks" from the other blocks. In this paper, two new algorithms, *ContentExtractor*, and *FeatureExtractor* are proposed. The algorithms identify primary content blocks by i) looking for blocks that do not occur a large number of times across webpages and ii) looking for blocks with desired features respectively. They identify the primary content blocks with high precision and recall, reduce the storage requirement for search engines, result in smaller indexes and thereby faster search times, and better user satisfaction. While operating on several thousand webpages obtained from 11 news websites, our algorithms significantly outperform the Entropy-based algorithm proposed by Lin and Ho [4] in both accuracy and run-time.

Keywords: Electronic Publishing, Data Mining, Information Systems Applications.

1. INTRODUCTION

Search engines crawl the World-Wide Web to collect webpages. These pages are stored and indexed. An end-user who performs a search using a search engine is interested in the *primary informative content* of the webpage. However, a substantial part of webpages is content that cannot be classified as the primary informative content of the webpage. These blocks are not relevant to the main content of the page and are seldom sought by the users of the website. We refer to such blocks as *non-content blocks*. Non-content blocks are very common in webpages. Typically such blocks contain advertisements, image-maps, plug-ins, logos, counters, search boxes, category information, navigational links, related links, footers and headers, and copyright information.

Before the content from a webpage can be used, it must be subdivided into smaller semantically-homogeneous components based on their content. We refer to such components as *blocks* in the rest of the paper. A block (or webpage block) $\mathcal{B}$ is a portion of a webpage enclosed within an open-tag and its matching close-tag, where the open and close tags belong to an ordered tag-set $\mathcal{T}$ that includes tags like <TR>, <P>, <HR>, and <UL>. Figure 1, shows a webpage obtained from CNN's website[1] and the blocks in that webpage. In this paper, we address the problem of identifying the primary informative content blocks of a webpage (e.g., in the figure, the text block containing the news is the primary content block).

An added advantage of identifying blocks in webpages is that if the user does not require the non-content blocks or requires only a few non-content blocks, we can delete the rest of the blocks. This contraction is useful in situations where large parts of the web are crawled, indexed and stored. Since the non-content blocks are often a significant part of webpages, eliminating them results in significant savings with respect to storage and indexing.

We propose simple yet powerful algorithms, called *ContentExtractor* and *FeatureExtractor*, to identify and separate content blocks from non-content blocks. We have characterized different types of blocks based on the different features they possess. *FeatureExtractor* is based on this characterization and uses heuristics based on the occurrence of certain features to identify content blocks. *ContentExtractor* identifies non-content blocks based on the appearance of the same block in multiple webpages.

First, the algorithms partition the webpage into blocks based on heuristics. Lin and Ho [4] have proposed an entropy-based algorithm that partitions a webpage into blocks on the basis of HTML tables. In contrast, not only do we consider HTML tables, but also other tags and use heuristics to partition a webpage. Second, our algorithms classifies each block as either a content block or a non-content block. While the algorithm decides whether a block, $\mathcal{B}$, is content or not, it also compares $\mathcal{B}$ with stored blocks to determine whether $\mathcal{B}$ is similar to a stored block. If $\mathcal{B}$ already exists in the repository, a second copy is not stored; instead a pointer to the

[1] http://www.cnn.com

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'05 March 13-17, 2005, Santa Fe, New Mexico, USA
Copyright 2005 ACM 1-58113-964-0/05/0003 ...$5.00.

Figure 1: A webpage from CNN.com and its blocks (shown using boxes)

stored block identical to $\mathcal{B}$ is retained.

Both *FeatureExtractor* and *ContentExtractor* produce excellent precision and recall values and above all, do not use any manual input and require no complex machine learning process. While operating on several thousand webpages obtained from 11 news websites, our algorithms significantly outperform their nearest competitor - the Entropy-based blocking algorithm proposed by Lin and Ho [4]. Although the problem of identifying content blocks is not exceedingly hard, apart from the *LH* algorithm there are no other direct candidates for comparisons.

The rest of the paper is organized as follows: In Section 2 we have discussed the related work. We describe our algorithms in Section 3. We outline our performance evaluation plan and the data set on which we ran our experiments in Section 4. Then, we compare our algorithms with the *LH* algorithm in Section 5 and conclude thereafter.

2. RELATED WORK

Yi and Liu [6, 5] have proposed an algorithm for identifying non-content blocks (the refer to it as "noisy" blocks) of webpages using style trees. Bar-Yossef and Rajagopalan [1] have proposed a method to identify frequent templates of webpages and pagelets (identical to our blocks). Kushmerick [3] has proposed a feature-based method that identifies internet advertisements in a web-page. The work that is most closely related to ours is by Lin and Ho [4].

The problem also has similarities with the problem of information extraction. Previous efforts have tried to automatically extract information that originally came from databases [2].

3. ALGORITHMS

We now discuss the two algorithms *ContentExtractor* and *FeatureExtractor*. The input to the algorithms is a set (at least two) of webpages belonging to a *class* of webpages. A class is defined as a set of webpages from the same website whose design or structural contents are very similar. A set of webpages with the same template is an example of a class. The output of the algorithms are the primary content blocks in the given class of webpages. The first step of both algorithms is to use the *GetBlockSet* routine. The *GetBlockSet* routine takes an HTML page as input with the ordered tag-set and partitions each page into blocks.

3.1 ContentExtractor

The ContentExtractor algorithm, shown in Algorithm 1 eliminates blocks depending upon the inverse block-document frequence, $IBDF$, of a block. The inverse block-document frequency, $IBDF$, is inversely proportional to the number of documents in which a block occurs or has a similar block. Blocks that are similar to blocks occuring in multiple pages in the same domain, e.g. blocks that occur in multiple pages at cnn.com, are identified as redundant blocks. Blocks that occur only in one page are identified as content-blocks.

Given two blocks, the similarity measure, SIM, returns the cosine between their block feature vectors. Examples of features are: the number of terms, the number of images, the number of java-scripts, etc. However, for text blocks, simply taking the number of terms in the block may result in falsely identifying two blocks as similar. Therefore, we augment the features by adding a binary feature for each term in the corpus of documents. If a feature occurs in a block, the entry in the corresponding feature vector is a one, otherwise it is zero. We used a threshold value of $\epsilon = 0.9$. That is, if the similarity measure is greater than the threshold value, then the two blocks are accepted as identical.

3.2 FeatureExtractor

The *FeatureExtractor* algorithm, shown in Algorithm 2, is invoked to identify blocks with a set of desired features. For example, *FeatureExtractor* invoked with the features text, image, links, identifies the text blocks, image blocks or navigational blocks. Then it partitions the set of blocks into two partitions using a clustering algorithm and selects the blocks that have the desired features.

3.2.1 Block Features

The following list describes the features of a webpage block that we have used in our implementation. A webpage block can have any or all features of an HTML page. The W3C HTML guidelines have been followed here.

Algorithm 1: *ContentExtractor*

Input : Set $\mathcal{S}$ of HTML pages, Sorted tag-set $\mathcal{T}$
Output: Primary Content Blocks and their associated pages in $\mathcal{S}$

begin
- $\mathcal{M}_{BD} \longleftarrow \emptyset$
- { Here the $\mathcal{M}_{BD}$ matrix is the block-document matrix where rows represent document and columns represent block identifier.}
- **for** *each* $H^k \in \mathcal{S}$ **do**
  - { Here B^k represents the kth row of the $\mathcal{M}_{BD}$ matrix. }
  - $B^k \longleftarrow$ **GetBlockSet**$(H^k, \mathcal{T})$
  - $\mathcal{M}^k_{BD} \longleftarrow B^k$
- **for** *each* $b_{ij} \in \mathcal{M}_{BD}$ **do**
  - $IBDF_{ij} \longleftarrow 1$
  - **for** *each* $b_{kl} \in \mathcal{M}_{BD}$ **do**
    - { Here $i \neq k$.}
    - $Sim_{ijkl} \longleftarrow SIM(b_{ij}, b_{kl})$
    - **if** $Sim_{ijkl} > \epsilon$ **then**
      - $IBDF_{ij} \longleftarrow Update(IBDF_{ij})$
      - {Update Recalculates IBDF}
- { If IBDF value above threshold we will produce the output }
- **for** *each* $b_{ij} \in \mathcal{M}_{BD}$ **do**
  - **if** $IBDF_i > \theta$ **then**
    - Output the content of the block

end

GetBlockSet : Input : HTML page H, Sorted tag-set $\mathcal{T}$
Output: Set of Blocks in H

begin
- $B \longleftarrow H$; // set of blocks, initially set to H. $f \longleftarrow Next(\mathcal{T})$
- **while** $f \neq \emptyset$ **do**
  - $b \longleftarrow First(B)$
  - **while** $b \neq \emptyset$ **do**
    - **if** b *contains* f **then**
      - $B^N \longleftarrow GetBlocks(B, f)$
      - $B \longleftarrow (B - b) \cup B^N$
    - $b \longleftarrow Next(B)$
  - $f \longleftarrow Next(\mathcal{T})$

end

- Text : The text content inside the block.
- Text-tag: The text tags, e.g., <h1>, <h2> etc. inside the block.
- List : The lists available inside the block.
- Style-Sheet : This is also to make the list complete and compliant to W3C guidelines. Styles are usually important for browser rendering, and usually included inside other tags, like links and tables etc.

We can update this list if we so desire because of changes in HTML features or because of an application's updated preferences of desirable features easily without fundamentally changing the algorithm.

Unlike the *ContentExtractor* algorithm, the *FeatureExtractor* algorit hm does not depend on multiple Web-pages but depends on the feature-set and the chosen feature for output. The set featu res are HTML features as explained before. For example, let us consider the chosen features is text ($\mathcal{T}_I$). Now our algorithm calculates a value for each feature in each block. Say, a block contains 1000 words and 2 images and 3 links and an applet, and the maximum values of words, images, links, and applets contained in blocks in the data-set are 2000, 4, 50 and 3. Then the values for the features in the given block are 1000/2000, 2/4, 3/50, and 1/3 respectively. After that we put each block in the winner-basket if the sum of the feature values of the desired features is greater than the sum of the feature values of the rest of the features. From this winner-basket, we recompute the feature values for this new set of blocks, and chose the one with highest sum of values of desired feature values.

Now according to this algorithm a block with a single word and nothing else woul d be theobvious winner and will be chosen. In all practical cases this scenario did not arise. And also, we do not consider a single row or column of a table as a block. We consider the whole table (in the highest depth of table tree) as a block. So the chance of getting a block with a single word is distant.

We extended the *FeatureExtractor* algorithm to choose 'k' blocks instead of a single block. We refer to the extended algorithm as *K-FeatureExtractor*. *K-FeatureExtractor* uses an adaptive K-means clustering on the winner set t o retrieve multiple winners as opposed to *FeatureExtractor* that selects a single winner.The usual values of k taken are 2 or 3, and the initial centroids are chosen from the sorted list at equidistant index values. After the clustering is done, the high probability cluster(s) are taken and the corresponding text contents of those blocks are output.

4. EVALUATION PLAN

We implemented and compared our algorithm with *LH*, the entropy-based algorithm, proposed by Lin and Ho [4]. They use the terms *precision* and *recall* to refer to the metrics to evaluate their algorithm. Although, the use of these terms are somewhat different from their usual sense in the "Information Retrieval" field, in order to avoid confusion, we use the same terms to refer to the evaluation metrics of our work.

4.1 Metric Used

Precision is defined as the ratio of the number of relevant items (actual primary content blocks) r found and the total

Algorithm 2: FeatureExtractor

Input : HTML pages H, Sorted Tag Set $\mathcal{F}$, Desired Feature $\mathcal{F}_I$

Output: Content Blocks of H

Feature: Feature set $\mathcal{F}_S$ used for block separation sorted according to importance taken from $\mathcal{F}$

begin

$B \longleftarrow GetBlockSet(H, \mathcal{F})$

for *each* $b \in B$ **do**

$P_1 \longleftarrow Pr(\mathcal{F}_I|\mathcal{F})$

if $P_1 > 0.5$ **then**

$\mathcal{W} \longleftarrow W \cup b$

Cluster $\mathcal{W}$ using k-means clustering with k=2.

{Output: Output high-probability cluster.}

end

number of items (primary content blocks suggested by an algorithm) t found. $Precision = \frac{r}{t}$. Recall has been defined as the ratio of the number of relevant items found and the desired number of relevant items. The desired number of relevant items includes the number of relevant items found and the missed relevant items m. $Recall = \frac{r}{r+m}$.

4.2 Data Set

Like Lin and Ho, we chose several websites from the news domain. We crawled the web for news articles and other types of websites to collect documents. The details (name, source, category, number) of the dataset are shown in Table 1.

We took 11 different news websites whose design and page-layouts are completely different. Unlike Lin and Ho's dataset [4] that is obtained from one fixed category of news sections (only one of them is "Miscellaneous" news from CDN), we took random news pages from every section of a particular website. This choice makes the dataset a good mix of a wide variety of HTML layouts. This step was necessary to compare the robustness of their algorithm to ours.

To compare the *FeatureExtractor* with *K-FeatureExtractor*, we constructed a new dataset consisting of a variety of websites.

5. PERFORMANCE COMPARISON

We implemented all three algorithms in Perl 5.8.0 on a Pentium-based Linux platform. With the help of a few graduate students and professors, we calculated the precision and recall values for each website and layout category for text feature. These values are shown in table 2.

Our algorithms outperform *LH* in all news sites in all categories. The recall is always good since both algorithms could find most relevant blocks but the results obtained by running the *LH* algorithm were less precise than those obtained by *ContentExtractor* since the former algorithm also includes lots of other non-content blocks.

5.1 Precision and Recall

Both *FeatureExtractor* and *ContentExtractor* performed better than *LH* in almost all cases.

We compare the features of all three algorithms in Table 4.

Site	Prec. of LH	Recall of LH	Prec. of ContentExtractor	Recall of ContentExtractor	Prec. of FeatureExtractor	Recall of FeatureExtractor
ABC	0.811	0.99	0.915	0.99	1.00	1.00
BB	0.882	0.99	0.997	1.00	1.00	1.00
BBC	0.834	0.99	0.968	1.00	1.00	1.00
CBS	0.823	1.00	0.972	1.00	0.98	0.977
CNN	0.856	1.00	0.977	1.00	0.98	0.98
FOX	0.82	1.00	0.967	1.00	1.00	0.99
FOX23	0.822	1.00	0.985	1.00	1.00	1.00
IE	0.77	0.95	0.911	0.993	0.93	0.99
IT	0.793	0.99	0.924	0.981	0.96	0.98
MSNBC	0.802	1.00	0.980	1.00	0.92	1.00
YAHOO	0.730	1.00	0.967	1.00	1.00	0.95

Table 2: Block level Precision and Recall values from *LH* algorithm, *ContentExtractor* and *FeatureExtractor*. The second and the third columns are from *LH* algorithm, the fourth and the fifth columns are from *ContentExtractor* and the sixth and seventh columns are from *FeatureExtractor*

The precision and recall values for both these algorithms are shown in table 3.

5.2 Execution Time

Figure 2 shows execution time taken by the three algorithms averaged over all test webpages.

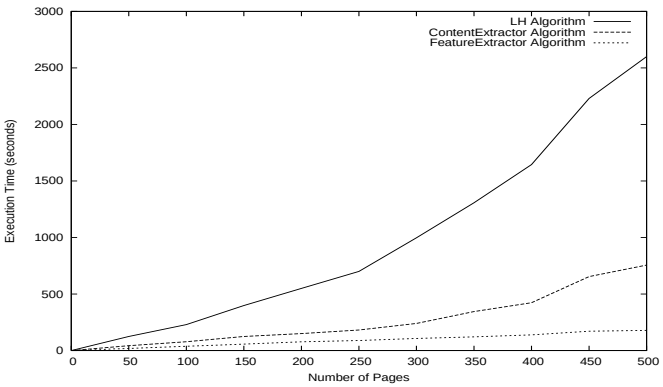

Figure 2: Run-times for the LH and the ContentExtractor algorithm.

In table 4 we present a comparison table for the features of both algorithms.

6. CONCLUSIONS AND FUTURE WORK

We devised simple, yet powerful, and modular algorithms, to identify primary content blocks from webpages. Our algorithms outperformed the LH algorithm significantly, in precision as well as run-time, without the use of any complex learning technique. The *FeatureExtractor* algorithm, provided a feature, can identify the primary content block with

Site	*Address*	*Category*	*Number*
ABC	http://www.abcnews.com	Main Page, USA, World, Business, Entertainment, Science/Tech, Politics,Living	415
BB	http://www.bloomsberg.com	Main Page, World, Market, US Top Stories, World Top Stories, Asian, Australia/New Zealand, Europe, The Americas	510
BBC	http://www.bbc.co.uk	Main Page, The Continents, Business, Health, Nature, Technology, Entertainment	890
CBS	http://www.cbsnews.com	Main Page, National, World, Politics, Technology, Health, Entertainment	370
CNN	http://www.cnn.com	Main Page, World, US, All Politics, Law, Tech(nology), Space (Technology), Health, Showbiz, Education, Specials	717
FOX	http://www.foxnews.com	Main Page, Top Stories, Politics, Business, Life, Views	476
FOX23	http://www.fox23news.com	Main Page, General, Local, Regional, National, World, In Depth, Sports, Business, Entertainment, Health	658
IE	http://www.indianexpress.com	Main Page, International, Sports, National Network, Business, Headlines	269
IT	http://www.indiatimes.com	Main Page, Main Stories, Top Media Headlines	454
MSNBC	http://www.msnbc.com	Main Page, Business, Sports, Technology an Science, Health, Travel	647
YAHOO	http://news.yahoo.com	Main Page, Top Stories, US (National), Business, World, Entertainment, Sports, Technology, Politics, Science	505

Table 1: Details of the dataset. Number of pages taken from individual categories are not shown due to the enormous size of the latex table. But interested reader can contact authors to get the details.

Site	**Pages**	**FEx. Pre.**	**FEx. Rec.**	**KFE. Pre.**	**KFE. Rec.**
ABC	258	1.00	1.00	1.00	1.00
BB	300	1.00	1.00	1.00	1.00
BBC	628	1.00	1.00	1.00	1.00
CBS	260	0.92	0.91	0.98	0.977
CNN	517	0.98	0.98	0.98	0.98
FOX	242	1.00	0.99	1.00	0.99
FOX23	458	1.00	1.00	1.00	1.00
IE	120	0.91	0.973	0.93	1.00
IT	354	0.96	0.85	0.96	0.85
MSNBC	347	0.91	0.99	0.92	1.00
YAHOO	405	1.00	0.95	1.00	0.95
Shopping	100	0.20	0.198	1.00	0.99
Amazon	100	0.33	0.32	1.00	0.967
ConsumerResearch	100	0.28	0.28	1.00	0.98
BarnesAndNoble	100	0.40	0.38	1.00	0.968
Epinion	100	0.5	0.478	1.00	0.956

Table 3: Details of the dataset and the block-level Precision and Recall values of the output produced by FeatureExtractor and K-FeatureExtractor compared.

Property	*LH*	*ContentExtractor*
Precision	Low	*High*
Recall	High	*Very High*
Number of pages needed	All the pages to calculate Entropy of features	*Very few* $(5-10)$ *pages from same class are enough to give high performance*
Time of completion	Always more than ContentExtractor	*Less than LH (shown in figure 2)*

Table 4: A property-wise comparison table for both algorithms

respect to that feature. The *ContentExtractor* algorithm detects redundant blocks based on the occurrence of the same block across multiple webpages. The algorithms, thereby, reduce the storage requirements, make indices smaller, and result in faster and more effective searches.

7. REFERENCES

[1] Z. Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. In *The Eleventh International World Wide Web Conference (WWW 2002)*. ACM Press, New York, NY, USA, 7-11 May 2002.

[2] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 109–118, 2001.

[3] Nicholas Kushmerick. Learning to remove internet advertisements. In *Third annual conference on Autonomous Agents*, pages 175–181. ACM Press, New York, NY, USA, 1999.

[4] S. Lin and J. Ho. Discovering informative content blocks from web documents. *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 588–593, 2002.

[5] L. Yi and B. Liu. Web page cleaning for web mining through feature weighting. In *Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, Aug 2003.

[6] L. Yi, B. Liu, and X. Li. Eliminating noisy information in web pages for data mining. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-2003)*, August 2003.