Identifying Content Blocks from Web Documents

Sandip Debnath[1], Prasenjit Mitra[1,2], and C. Lee Giles[1,2]

[1] Department of Computer Science and Engineering,
[2] School of Information Sciences and Technology,
Penn State University, University Park, PA 16802, USA
debnath@cse.psu.edu, {pmitra, giles}@ist.psu.edu

Abstract. Intelligent information processing systems, such as digital libraries or search engines index web-pages according to their informative content. However, web-pages contain several non-informative contents, e.g., navigation sidebars, advertisements, copyright notices, etc. It is very important to separate the informative "primary content blocks" from these non-informative blocks. In this paper, two algorithms, *FeatureExtractor* and *K-FeatureExtractor* are proposed to identify the "primary content blocks" based on their features. None of these algorithms require any supervised learning, but still can identify the "primary content blocks" with high precision and recall. While operating on several thousand web-pages obtained from 15 different websites, our algorithms significantly outperform the Entropy-based algorithm proposed by Lin and Ho [14] in both precision and run-time.

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

1 Introduction

An end-user is mainly interested in the *primary informative content* of these web-pages. However, a substantial part of web-pages is not very informative in nature. So these parts or blocks (defined later) are seldom sought by the users. We refer to such blocks as *non-content blocks* which includes advertisement blocks, image-maps, plug-ins, logos, counters, search boxes, category information, navigational links, related links, footers and headers, and copyright information among others. In this paper, we address the problem of identifying the primary informative content of a web-page. An added advantage of this is that after identifying all the blocks, we can delete the non-content 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 [1] of dynamically generated web-pages, eliminating them results in significant savings with respect to storage and

[1] sometimes the non-content blocks' total size is as much as 65-70% of the whole page-size.

M.-S. Hacid et al. (Eds.): ISMIS 2005, LNAI 3488, pp. 285–293, 2005.

index file-size. We have designed and implemented two algorithms, *FeatureExtractor*, and *K-FeatureExtractor*, which can identify the primary content blocks in a web-page. First, the algorithms partition the web-page into blocks based on heuristics. Lin and Ho [14] have proposed an entropy-based algorithm that partitions a web-page into blocks on the basis of HTML tables. In contrast, not only do we consider HTML tables, but also other tags and heuristics to partition a web-page. Secondly, our algorithms classifies each block as either a content block or a non-content block. Both **FeatureExtractor** and **K-FeatureExtractor**) 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 web-pages obtained from 15 news websites, our algorithms significantly outperform their nearest competitor - the Entropy-based blocking algorithm proposed by Lin and Ho [14].

The rest of the paper is organized as follows. In section 2 we have discussed the related work. We define the concept of "blocks" and a few related terms in section 3, describe our algorithms in sections 4 and 5, and outline our performance evaluation plan and the dataset in section 6. We then compare our algorithms with the **LH** algorithm in section 7 and conclude thereafter.

2 Related Work

Yi and Liu [17, 15] have proposed an algorithm for identifying non-content blocks of web-pages using "Style Tree". Since our algorithms use simple heuristics to determine non-content blocks, it does not incur the overhead of constructing "Style Tree"s. Another work that is closely related is the work by Lin and Ho [14]. Their algorithm tries to partition a web-page into blocks and identify content blocks. They used the entropy of the keywords used in a block to determine whether the block is redundant. We believe that we have a more comprehensive definition of blocks and demonstrate that we have designed and implemented an algorithm that gives better precision and recall values than their algorithm. Bar-Yossef and Rajagopalan [3] have proposed a method to identify frequent templates of web-pages and pagelets (identical to our blocks). Yi and Liu argue that their entropy-based method supersedes the template identification method. We show that our method produces better result than the entropy-based method. Kushmerick [12, 11] has proposed a feature-based method that identifies Internet advertisements in a web-page. Their algorithm generates rules from training examples using a manually-specified procedure that states how the features to be used can be identified. This manual specification is dependent upon applications. Our algorithms do not require any manual specification or training data set.

Information extraction systems try to extract useful information from either structured, or semi-structured documents. Systems like Tsimmis [4] and Araneus [2] depend on manually provided grammar rules. In Information Manifold [10, 13], Whirl [6], or Ariadne [1], the systems tried to extract information using a query system that is similar to database systems. In Wrapper systems [12], the wrappers are automatically created without the use of hand-coding. Kushm-

erick et. al. [12, 11] have found an inductive learning technique. Their algorithm learns a resource's wrapper by reasoning about a sample of the resource's pages. In Roadrunner [7], a subclass of regular expression grammar (UFRE or Union Free Regular Expression) is used. In Softmealy [9], a novel web wrapper representation formalism has been presented based on a finite-state transducer (FST) and contextual rules. For other semi-structured wrapper generators like Stalker [16], a hierarchical information-extraction technique converts the complexity of mining into a series of simpler extraction tasks. Most of these approaches are geared toward learning the regular expressions or grammar induction [5] of the inherent structure or the semi-structure and so computational complexities are quite high.

These efforts are to extract information that originally comes from databases, which is very structured in nature. Our work concentrates on web-pages where the underlying information is unstructured text. Our preliminary work [8] shows great improvements in extracting informative blocks from web-pages. We described our **ContentExtractor** algorithm in [8] which uses multiple web-pages from same source and find repetitive similar blocks. By eliminating these repetitive blocks it improved the precision and recall of finding content blocks. In [8] we mentioned **FeatureExtractor** algorithm very briefly. Here we introduce **K-FeatureExtractor** algorithm and to describe it properly, we believe that an introduction of **FeatureExtractor** was necessary. **K-FeatureExtractor** even outperform **ContentExtractor** in runtime.

3 Blocks in Web Pages

A block (or web-page block) $\mathcal{B}$ is a portion of a web-page 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>.

Heuristics
Out of all these tags, web authors extensively use <TABLE> for layout design. We devised a list of tags to partition a web-page into blocks. <TABLE> comes as the first or tag in that list. <TR>, <P>, <HR>, and <UL> etc. are the next few partitioning tags in that list, in order. We selected the order of the tags based on our observations of web-page design. For example, <TABLE> comes as the first partitioning tag since we see more instances of <UL> in <TR> / <TD> (sub-element of <TABLE>) than <TABLE>s coming inside <LI> (sub-element of <UL>). Our algorithms partition a web-page into blocks, based on the first tag in our list. It continues sub-partitioning the already-identified blocks based on the next listed tags. The partitioning algorithm is illustrated in next section.

Block Features
Blocks may include other smaller blocks, and have features like text, images, applets, javascript, etc. Most, but not all, features are associated with their respective standard tags. For example, an image is always associated with the

tag <IMG>, however, the text feature has no standard tag. For tag-associated features we used the W3C [2] guidelines to make the list of features. We can update this list as time and version of HTML pages change, without doing any fundamental change in our algorithms.

4 Algorithm: FeatureExtractor

We describe our algorithms **FeatureExtractor** and **K-FeatureExtarctor** here. As some parts of these two algorithms are similar, we show them both in Algorithm 1 to save space.

FeatureExtractor takes an HTML page, a desired feature and a sorted tag set (for block-partitioning purpose). It first partitions the page into blocks with the help of *GetBlockSet* routine. It then calculates the probability of individual blocks for the desired feature. It takes those blocks in the winner set for which the probability of the desired feature is more than the combined probability of the rest of the feature. In the next step, within the winner set, it finds the block with the highest probability value and extracts the information (in case of text feature it extracts the text part) from the block.

GetBlockSet
GetBlockSet takes a tag from the tag-set one by one and calls the *GetBlocks* routine for each of the already-generated blocks. New sub-blocks created by *GetBlocks* are added to the block set and the generating main block is removed from the set. $First$ gives the first element of an ordered set, and $Next$ gives the consecutive elements.

GetBlocks
GetBlocks takes a full or part of an HTML document, and a tag as its input. It partitions the document into blocks according to the input tag. For example, for <TABLE> input tag it will produce a tree with all the table blocks.

5 Algorithm: K-FeatureExtractor

FeatureExtractor shows high precision and recall for most of the websites in our dataset. However, for web-pages with multiple important text blocks, a typical reader may be interested in all of them, instead of just the winner block (from **FeatureExtractor**). General shopping sites, review sites, chat forums etc. comes under this category. Undoubtedly **FeatureExtractor**, shows poor performance. To overcome this we improved **FeatureExtractor**. This improved algorithm, **K-FeatureExtractor**, is also shown in algorithm 1. Here, instead of taking just the winner from the winner-basket, we apply a K-means clus-

[2] World Wide Web Consortium or http://www.w3c.org/TR/html4/

Algorithm 1: (K)-FeatureExtractor

Input : Set of HTML pages H, Sorted Tag Set $\mathcal{T}$, Desired Feature $\mathcal{F}_I$
Output: Content Block(s) of H and its content. It returns the contents as well as identifications of the blocks to assist the block-level calculation of b-Precision and b-Recall
Feature: Feature set $\mathcal{F}_S$ used for block separation sorted according to importance taken from $\mathcal{T}$

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**
      $\mathcal{W} \longleftarrow W \cup b$
  for *each* $b \in \mathcal{W}$ **do**
    $P_b \longleftarrow Pr(\mathcal{F}_I|\mathcal{F}, \mathcal{W})$
  //In case of FeatureExtractor we used the following method
  $W_s \longleftarrow Sort(\mathcal{W})$
  //Winner block
  $B_w \longleftarrow First(W_s)$
  $C \longleftarrow ContentOfBlock(B_w)$
  Return (B_w, C)
  //In case of K-FeatureExtractor we used the following method
  $W_s \longleftarrow KMeansClustering(\mathcal{W})$
  //Winner blockset
  ${W_s}^w \longleftarrow FirstCluster(W_s)$
  $C \leftarrow \phi$
  for *each* ${B_w} \in {W_s}^w$ **do**
    $C \longleftarrow C \cup ContentOfBlock(B_w)$
  Return (W_s, C)

  Function: 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
end

tering to select the best probability block-cluster from the preliminary winning basket. After the clustering is done, the high probability cluster is taken and the corresponding *text* contents of all those blocks are combined as the output (the desired feature is *text* here). Both results from **K-FeatureExtractor** and **FeatureExtractor** are shown in table 2.

6 Evaluation Plan

Lin and Ho [14] used *precision* and *recall* to evaluate their algorithm. Although it is confusing and somewhat different from their usual application in "Information Retrieval", we use the same terms (added with a "b-" for blocks) (in order to avoid confusion).

Metric Used
Precision is defined as the ratio of the number of relevant items (actual primary content blocks) r found and the total number of items (primary content blocks suggested by an algorithm) t found. For block level precision, we call it as $b - Precision$. $b - Precision = \frac{r}{t}$. Recall is defined as the ratio of the number of relevant items found and the desired number of relevant items. The number of missed relevant items is m. In case of blocks we call it as $b - Recall$. $b - Recall = \frac{r}{r+m}$. We define the F-measure here as b-F-measure and define it as $b - F - measure = \frac{2*(b-Precision)*(b-Recall)}{(b-Precision)+(b-Recall)}$

Table 1. Details of the dataset. Categories are not shown due to the enormous size of the latex table. But interested reader can find the categories in [8]

Site	Web-address	Number of Web-pages
ABC	http://www.abcnews.com	415
BB	http://www.bloomsberg.com	510
BBC	http://www.bbc.co.uk	890
CBS	http://www.cbsnews.com	370
CNN	http://www.cnn.com	717
FOX	http://www.foxnews.com	476
FOX23	http://www.fox23news.com	658
IE	http://www.indianexpress.com	269
IT	http://www.indiatimes.com	454
MSNBC	http://www.msnbc.com	647
YAHOO	http://news.yahoo.com	505
Shopping	http://www.shopping.com	100
Amazon	http://www.amazon.com	100
Barnes And Noble	http://www.bn.com	100
Epinion	http://www.epinions.com	100

Data Set
Similar to Lin and Ho, we chose several news websites. We also chose shopping and book websites. We crawled these sites to collect documents. The details of the dataset are shown in Table 1 and in [8].

We took 15 different websites whose design and page-layouts are completely different. Unlike Lin and Ho's dataset [14] 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 a particular website (details in [8]). This makes the dataset a good mix of a wide variety of HTML layouts which is necessary to compare the robustness of **LH** algorithm and ours

7 Performance Comparison

b-Precision and b-Recall
The b-precision and b-recall values for each website are shown in table 2. Our algorithms outperform **LH** in all cases. The results from **LH** algorithm are less precise than those obtained by **K-FeatureExtractor**.

Execution Time
Figure 1 shows execution time taken by algorithms (*LH*, and *(K)-FeatureExtractor*) averaged over all test web-pages. From the figure it is clear that our algorithms outperform the *LH* algorithm by huge margin.

Table 2. Block level Precision and Recall values from *LH* algorithm, *FeatureExtractor*, and *K-FeatureExtractor*. For cases where these results are same for *FeatureExtractor* and *K-FeatureExtractor* we mentioned it once

Site	**b-Prec of LH**	**b-Recall of LH**	**b-F-measure of LH**	**b-Prec of FE/K-FE**	**b-Recall of FE/K-FE**	**b-F-measure of FE/K-FE**
ABC	0.811	0.99	**0.89**	1.00	1.00	**1.00**
BB	0.882	0.99	**0.93**	1.00	1.00	**1.00**
BBC	0.834	0.99	**0.905**	1.00	1.00	**1.00**
CBS	0.823	1.00	**0.902**	0.98	0.977	**0.978**
CNN	0.856	1.00	**0.922**	0.98	0.98	**0.98**
FOX	0.82	1.00	**0.901**	1.00	0.99/1.00	**0.994/1.00**
FOX23	0.822	1.00	**0.902**	1.00	1.00	**1.00**
IE	0.77	0.95	**0.85**	0.93	0.99	**0.959**
IT	0.793	0.99	**0.878**	0.96	0.98	**0.969**
MSNBC	0.802	1.00	**0.89**	0.92	1.00	**0.95**
YAHOO	0.730	1.00	**0.84**	1.00	0.95	**0.974**
Shopping	0.79	1.00	**0.88**	1.00	0.25/0.99	**0.4/0.994**
Amazon	0.771	0.99	**0.86**	1.00	0.35/0.967	**0.51/0.983**
Barnes And Noble	0.81	1.00	**0.895**	1.00	0.34/0.968	**0.50/0.983**
Epinion	0.79	1.00	**0.88**	1.00	0.289/0.956	**0.45/0.977**

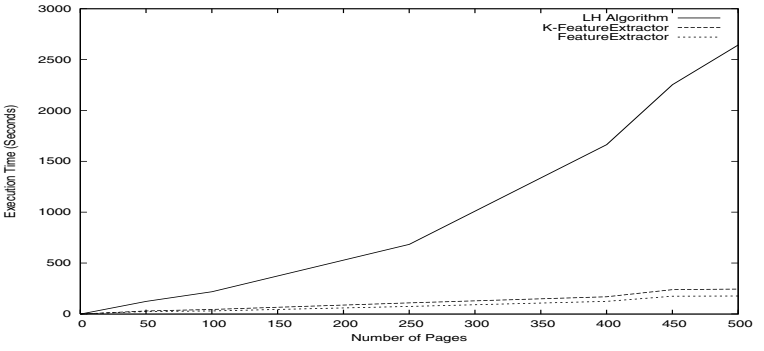

Fig. 1. Run-times for the **LH**, **FeatureExtractor**, and **K-FeatureExtractor** algorithms. The vertical axis represents the time of execution (in seconds) for a number of pages (plotted in the horizontal axis). It is clear that our algorithms outperform the **LH** algorithm in large margin. **K-FeatureExtractor** even outperforms **ContentExtractor** [8]

8 Conclusions and Future Work

We devised simple, yet powerful, and modular algorithms, to identify primary content blocks from web-pages. Our algorithms outperformed the LH algorithm significantly, in b-precision and run-time. In the next step, we will try to identify the semantics of the content to generate markup. The storage requirement for indices, the efficiency of the markup algorithms, and the relevancy measures of documents should also improve since now only the relevant parts of the documents are considered.

Acknowledgements

We acknowledge the help from several graduate students at Penn State University, specially from Pradeep B. Teregowda, and Isaac Councill.

References

1. José Luis Ambite, Naveen Ashish, Greg Barish, Craig A. Knoblock, Steven Minton, Pragnesh J. Modi, Ion Muslea, Andrew Philpot, and Sheila Tejada. Ariadne: a system for constructing mediators for Internet sources. In *SIGMOD*, pages 561–563, 1998.
2. Paolo Atzeni, Giansalvadore Mecca, and Paolo Merialdo. Semistructured and structured data in the web: Going back and forth. In *Workshop on Management of Semistructured Data*, 1997.
3. Ziv Bar-Yossef and Sridhar Rajagopalan. Template detection via data mining and its applications. In *Proceedings of WWW 2002*, pages 580–591, 2002.

4. Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yanis Papakonstantinon, Jeffrey Ullman, and Jennifer Widom. The tsimmis project: integration of heterogeneous information sources. In *Proceedings of the 10th meeting og Information Processing Society of Japan*, pages 7–18, 1994.
5. Boris Chidlovskii, Jon Ragetli, and Maarten de Rijke. Wrapper generation via grammar induction. In *Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain, May 31 - June 2, 2000, Proceedings*, volume 1810, pages 96–108. Springer, Berlin, 2000.
6. William W. Cohen. A web-based information system that reasons with structured collections of text. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 400–407, New York, 9–13, 1998. ACM Press.
7. Valter Crescenzi, Giansalvatore Mecca, and Paolo 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.
8. Sandip Debnath, Prasenjit Mitra, and C. Lee Giles. Automatic extraction of informative blocks from webpages. In *the upcoming proceedings of the Special Track on Web Technologies and Applications in the ACM Symposium of Applied Computing*, 2005.
9. C. Hsu. Initial results on wrapping semistructured web pages with finite-state transducers and contextual rules. In *AAAI-98 Workshop on AI and Information Integration*, pages 66–73. AAAI Press, 1998.
10. Thomas Kirk, Alon Y. Levy, Y Sagiv, and Divesh Srivastava. The information manifold. In *Proceedings of the AAAI Spring Symposium: Information Gathering from Heterogeneous Distributed Environments*, pages 85–91, 1995.
11. Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.
12. Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.
13. Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems - Special Issue on Networked Information Discovery and Retrieval*, 5(2):121–143, 1995.
14. Shian-Hua Lin and Jan-Ming Ho. Discovering informative content blocks from web documents. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 588–593, 2002.
15. Bing Liu, Kaidi Zhao, and Lan Yi. Eliminating noisy information in web pages for data mining. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 296–305, 2003.
16. Ion Muslea, Steven Minton, and Craig A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.
17. Lan Yi, Bing Liu, and Xiaoli Li. Visualizing web site comparisons. In *Proceedings of the eleventh international conference on World Wide Web*, pages 693–703, 2002.