Page-level Template Detection via Isotonic Smoothing

Deepayan Chakrabarti
Yahoo! Research
701 First Ave
Sunnyvale, CA 94089.
deepay@yahoo-inc.com

Ravi Kumar
Yahoo! Research
701 First Ave
Sunnyvale, CA 94089.
ravikumar@yahoo-inc.com

Kunal Punera*
Dept of ECE
Univ. Texas at Austin
Austin, TX 78712.
kunal@ece.utexas.edu

ABSTRACT

We develop a novel framework for the page-level template detection problem. Our framework is built on two main ideas. The first is the automatic generation of training data for a classifier that, given a page, assigns a templateness score to every DOM node of the page. The second is the global smoothing of these per-node classifier scores by solving a regularized isotonic regression problem; the latter follows from a simple yet powerful abstraction of templateness on a page. Our extensive experiments on human-labeled test data show that our approach detects templates effectively.

Categories and Subject Descriptors

H.4.m [**Information Systems**]: Miscellaneous

General Terms

Algorithms, Experimentation, Measurements

Keywords

Webpage sectioning, webpage segmentation, template detection, isotonic regression

1. INTRODUCTION

The increased use of content-management systems to generate webpages has significantly enriched the browsing experience of end users; the multitude of site navigation links, sidebars, copyright notices, and timestamps provide easy-to-access and often useful information to the users. From an objective standpoint, however, these "template" structures pollute the content by digressing from the main topic of discourse of the webpage. Furthermore, they can cripple the performance of many modules of search engines, including the index, ranking function, summarization, duplicate detection, etc. With templated content currently constituting more than half of all HTML on the web and growing steadily [3, 11], it is imperative that search engines develop scalable tools and techniques to reliably detect templates on a webpage.

Most existing methods for template detection operate on a per website basis by analyzing several webpages from the site and identifying content and/or structure that repeats across many pages. While these "site-level" template detection methods offer a lot of promise, they are of limited use because of the following two reasons. First, site-level templates constitute only a small fraction of all templates on the web. For instance, page- and session-specific navigation aids such as "Also bought" lists, ads, etc. are not captured by the site-level notion of templates. Second, these methods are error prone when the number of pages analyzed from a site is statistically insignificant, either because the site is small, or because a large fraction of the site is yet to be crawled. In particular, they are totally inapplicable when pages from a new website are encountered for the first time. An alternative paradigm that avoids many of these pitfalls is to detect templates on per webpage basis, i.e., "page-level" template detection. This is especially attractive since it can be easily deployed as a drop-in module in existing crawler work-flows.

A tempting approach to page-level template detection is to extract sufficiently rich features from the DOM nodes and train a classifier to assign "templateness" scores to each node in a DOM tree. While this approach is entirely plausible, it has several handicaps. First, for the classifier to have a reasonable performance, both accurate and comprehensive training data is required; this can involve prohibitive human effort. Second, by classifying each DOM node in isolation this approach does not take a global view of the templateness of nodes in the DOM tree.

In this paper we develop a novel framework for page-level template detection.

Main contributions. Our first contribution is a method to automatically build a page-level templateness classifier. This method works as follows. First, we generate training data by applying the site-level template detection method on several randomly selected sites. Next, we define and extract appropriate features for these site-level templates. Finally, we use this automatically generated data to train a classifier that can assign a templateness score to every node in a DOM tree. We show that our classifier generalizes beyond its site-level training data and can also discover templates that manifest only at the page-level.

Our second contribution is the formulation of a global property that relates templateness scores across nodes of the DOM tree. We assert that templateness is a monotone property: a node in the DOM tree is a template if and only if all its children are templates. An appropriate relaxation of this property leads to the following regularized isotonic regression problem: given a tree with classifier scores at each

*Most of the work was done while the author was visiting Yahoo! Research.

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

node, find smoothed scores that are not far from the classifier scores, but satisfy the relaxed monotonicity property. We provide an efficient algorithm to optimally solve this problem; this algorithm may be of independent interest.

On the whole, our approach eliminates the aforementioned issues with pure classifier-based approaches. An interesting by-product of our framework is that we obtain a sectioning of a page into segments; this is useful in many applications.

We perform an extensive set of experiments to validate our framework and algorithm. In terms of detecting content within templates, our algorithm achieves an f-measure in excess of 0.65 for text and 0.75 for links on a human-labeled test set. We highlight the applications of template detection by showing that removing templates as a pre-processing step boosts the accuracy of standard web mining tasks on our datasets, by as much as 140% on duplicate detection, and 18% on webpage classification. Further, we show that the gains obtained by using our page-level template detection approach are substantially greater than those obtained by using the more expensive site-level approach.

2. RELATED WORK

Our work in this paper is related to two broad areas of research: template detection and isotonic regression.

Site-level template detection. The problem of template detection and removal was first studied by Bar-Yossef and Rajagopalan [3], who proposed a technique based on segmentation of the DOM tree, followed by the selection of certain segments as candidate templates depending on their content. Yi et al. [26] and Yi and Liu [25] used a data structure called the *style tree* to take into account the metadata for each node, instead of its content. Vieira et al. [24] framed the template detection problem as a problem of mapping identical nodes and subtrees in the DOM trees of two different pages. They proposed performing the expensive task of template detection on a small number of pages, and then removing all instances of these templates from the entire site by a much cheaper approach. Gibson et al. [11] conducted a detailed study of templates on the web, demonstrating the prevalence of templated content and its steady growth.

All of these methods, however, need multiple pages from the same website to perform template detection, and thus suffer from the problems mentioned in the introduction. Our page-level algorithm can *use* the site-level templates detected by these methods as training data, generalizing the concept of a template beyond what is found by these algorithms.

Page-level template detection. Some page-level algorithms have also been proposed recently. Kao et al. [14] segment a given webpage using a greedy algorithm operating on features derived from the page. However, their method is not completely page-level; they also use some site-level features such as the number of links between pages on a website. Debnath et al. [10] also propose a page-level algorithm ("L-Extractor") that applies a classifier to DOM nodes (as in our algorithm), but only certain nodes are chosen for classification, based on a predefined set of tags. Kao et al. [13] propose a scheme based on information entropy to focus on the links and pages that are most information-rich, reducing the weights of template material as a by-product. Song et al. [22] use visual layout features of the webpage to segment it into blocks which are then judged on their salience and quality. Other local algorithms based on machine learning have been proposed to remove certain types of template material. Davison [9] uses decision tree learning to detect and remove "nepotistic" links, and Kushmerick [16] develops a browsing assistant that learns to automatically removes banner advertisements from pages. Another set of papers focus only on segmentation of webpages for the purpose of displaying them on small mobile device screens [2, 7, 27]. However, template detection is not their primary focus.

While similar in spirit to our page-level template detection system, these algorithms are significantly different in the details. Only a subset of DOM nodes ("segments") are operated upon, this subset having been chosen prior to any determination of the templateness of those segments. As a result, should a segment itself be composed of several template and non-template nodes, this would not be detected. Our algorithm operates on each node in the DOM tree, and finds segments based on the results of a templateness classifier; thus, it avoids these problems.

Isotonic regression. The problem of isotonic regression crops up in a range of disciplines, from microarray data analysis [1] to epidemiology [19] and statistics [21]. Stout [23] showed that the optimal solution for complete orders can be computed in $O(n \log n)$ for L_1, and $O(n)$ time for L_2 distance metrics. Pardalos and Xue [20] gave an $O(n \log n)$ algorithm for L_2 isotonic regression on rooted trees. Angelov et al. [1] recently presented an $O(n^2 \log n)$ algorithm for L_1 on trees.

We propose a general version of the isotonic regression problem on trees that subsumes the problems mentioned above. For the L_1 distance metric we provide an exact algorithm to solve this general problem in $O(n^2 \log n)$ time, matching the best results on the special cases studied in the results cited above.

3. FRAMEWORK

In this section we describe the proposed framework for the page-level template detection problem. We first fix some notation.

Recall the DOM tree representation of an HTML document where each node in the DOM tree corresponds to an HTML fragment; we identify the DOM node with the fragment it represents. Let T be the rooted DOM tree corresponding to the document. From here onwards, we use the tree T as a metaphor for the document. Let $\text{templ}(T)$ denote the set of all nodes in T that are templates. We use $i \in T$ to denote that node i belongs to tree T, $\text{parent}(i)$ to denote the parent of i in T, $\text{child}(i)$ to denote the set of children of i in T, and $\text{root}(T)$ to denote the root of T.

Let $\mathcal{H}$ denote the set of all possible DOM nodes. In the page-level template detection problem, we seek a boolean function $\tau : \mathcal{H} \to \{0, 1\}$ such that $\tau(i) = 1$ for all $i \in \text{templ}(T)$, and $\tau(i) = 0$ otherwise. In a relaxed version of the problem, we seek a function $\tilde{\tau} : \mathcal{H} \to [0, 1]$ where if $i \in \text{templ}(T)$ and $j \notin \text{templ}(T)$, then $\tilde{\tau}(i) > \tilde{\tau}(j)$; using an appropriate threshold, we can round $\tilde{\tau}$ to make it boolean.

A first-cut approach to page-level template detection would be to extract sufficiently rich features from the DOM nodes (in the context of a page) and train a classifier $x : \mathcal{H} \to [0, 1]$ to score the "templateness" of each node in a given page. While this appears plausible, it has several issues when scrutinized closely. The first set of issues revolve around the con-

struction of the training data for our classifier. For the classifier to learn the notion of "templateness" of DOM nodes on the web in general, it must be trained comprehensively over all forms of templates that it is likely to encounter. The heterogeneity and scale of the web imply that a huge corpus of accurate and diverse training data will be required. These requirements present a daunting task that demands tremendous human effort. Secondly, this approach to classification ignores the global property of templateness in the DOM tree, crisply stated as follows.

PROPERTY 1 (TEMPLATENESS MONOTONICITY). *A node in the tree is a template if and only if all its children are templates. In other words, the function $\tau(\cdot)$ is monotone on the tree.*

As is apparent, by working on each node of T in isolation, a naive classifier misses this intuitive relationship among templateness of nodes in the tree.

Our three step framework is meant precisely to address these issues, and is described below.

1. Automatic generation of training data. The first step is the automatic generation of training data. To this end, we use the site-level template detection paradigm of [11]. Note that even though site-level template detection is less feasible as a web-scale template detection mechanism, we show that we can still use it to generate training data for our approach.

The basic intuition behind the site-level template detection approach is the following. One of the common properties of templates is that they occur repeatedly across many pages on a single site. Therefore, if a DOM node occurs many times on different pages from a single site, then it lends credible evidence that this DOM node perhaps corresponds to a template.

We now describe a generic algorithm that we will call SITELEVEL (θ). This algorithm operates on a site by site basis. For each site, it obtains a set $\mathcal{T}$ of random pages from the site. Then, for each page $T \in \mathcal{T}$ and for every DOM node $i \in T$, it computes $h(i)$, where $h(\cdot)$ is a random hash function. Let $I_\theta^+ \subseteq \mathcal{H}$ be the set of DOM nodes that occur on at least θ fraction of pages in $\mathcal{T}$. Note that using hashes, this set can be identified efficiently. SITELEVEL returns I_θ^+ as the set of DOM nodes deemed templates.

2. Classification. The second step is to use the set of DOM nodes I_θ^+ identified by SITELEVEL as training data for a classifier. For this, we first identify appropriate features of DOM nodes in I_θ^+, in the context of the pages they appear in. We then train a classifier $x : \mathcal{H} \to [0, 1]$ using these features of the DOM nodes, treating those in I_θ^+ as positive examples; the output of the classifier is a templateness score for a given DOM node in a tree. The hope in using a classifier is that it can distill features from site-level templates that can be generalized to all templates on the web. This can help us identify templates that don't manifest themselves by repeatedly occurring across multiple pages on a website — templates that a pure site-level template detection approach cannot discover by itself. As we empirically observe in Section 6.2, this is indeed what happens.

3. Isotonic smoothing. At this point, one could use the classifier to assign a templateness score $x(\cdot)$ to each DOM node in the given page T. However, as we argued earlier, this does not fully capture the essence of the problem since the templateness scores assigned by the classifier to each DOM node in isolation may not satisfy the property that a node is a template if and only if all its children are templates (Property 1). On the other hand, assuming the classifier has reasonable accuracy, the scores it assigns makes sense for most, if not for all, of the nodes. The question now is how to reconcile the score assigned by the classifier with the monotonicity property of the templates.

To handle this question, we first consider a natural generalization of the monotonicity property for the case of real valued templateness scores. Suppose $y(i)$ is the templateness score of a node i in the tree. Then, $y(\cdot)$ is said to satisfy *generalized templateness monotonicity* if for every internal node i, with children $j_1, \ldots, j_\ell$, $y(i) = \min\{y(j_1), \ldots, y(j_\ell)\}$, i.e., the templateness of an internal node is the equal to the least of its children's templateness scores.

Note that generalized monotonicity ensures, first, that the templateness score of a node is at least the templateness score of its parent, and second, that the templateness score of the parent equals the templateness score of all its children, when the children all have same templateness score. We also have an additional requirement that the templateness score $y(\cdot)$ be close to the $x(\cdot)$ scores assigned by the classifier. Generalized monotonicity together with this closeness requirement leads to the problem of generalized isotonic regression on trees, which we solve in this paper.

While we defer the detailed description of our solution to the next section, we now highlight the advantages of our framework. Besides addressing the issues with using just the classifier scores, our framework offers additional benefits. First, the overall framework in simple and modular. Second, any off-the-shelf classifier can be used, instead of having to design one that works specifically for the given DOM tree structure. Third, as we will see later, a neat by-product of isotonic smoothing is that we obtain a sectioning of a page into segments; this can be useful in many applications.

4. ISOTONIC SMOOTHING

In this section we formulate and solve the generalized isotonic regression problem on trees. Recall that we are given as input a DOM tree with each node labeled by a score assigned by the classifier. The purpose of isotonic regression is to fix these scores so that they satisfy the monotonicity constraints, while remaining as faithful as possible to the original classifier scores. Let $x(i)$ be the classifier score for each node $i \in T$ and let $y(i)$ be the smoothed score we wish to obtain.

The first step in our formulation is to alter the generalized monotonicity property in two ways. First, we only ensure that the templateness score of a node is *at most* the least of its children's scores, instead of *equal* to it. This relaxation is derived from the current domain in which the cost of misclassifying a non-template as a template is much higher than vice versa. Hence, if according to the classifier an internal node's template score is much lower than that of all of its children, then we would want to respect that. Second, we introduce a regularization that penalizes if, for a node i, the templateness score $y(i)$ is different from those of its children $y(j_1), \ldots, y(j_k)$. Clearly, if $y(j_1) = \cdots = y(j_k)$, then this regularization will try to ensure that $y(i) = y(j_1)$.

Thus, we have

(1) For every internal node i with children $j_1, \ldots, j_\ell$, $y(i) \leq \min\{y(j_1), \ldots, y(j_\ell)\}$.

For purposes of regularization, we develop the notion of compressed score that embodies sectioning of the DOM tree into subtrees. A *compressed score* is a function $\hat{y} : T \to [0,1] \cup \{\perp\}$ with the following properties:

(2) $\hat{y}(\text{root}(T)) \neq \perp$, and

(3) if i is an ancestor of j and $\hat{y}(i) \neq \perp \neq \hat{y}(j)$, then $\hat{y}(i) < \hat{y}(j)$.

Let the *size* $|\hat{y}|$ of the compressed score be the number of places where $\hat{y}$ is defined; $|\hat{y}| = |\{i \mid i \in T, \hat{y}(i) \neq \perp\}|$.

For all $i \in T$ such that $\hat{y}(i) = \perp$, let $\text{anc}(i)$ be the closest ancestor of i such that $\hat{y}(\text{anc}(i)) \neq \perp$; note that such an ancestor always exists by (2). We now interpolate $\hat{y}$ to a unique y as follows.

$$y(i) = \begin{cases} \hat{y}(\text{anc}(i)) & \text{if } \hat{y}(i) = \perp \\ \hat{y}(i) & \text{otherwise} \end{cases}$$

It is clear that if $\hat{y}$ satisfies (2) and (3), then the corresponding interpolated y satisfies (1). Also, given a y satisfying (1), it is easy to construct the unique $\hat{y}$. From now on, we use the smoothed score y and its compressed counterpart $\hat{y}$ interchangeably.

Finally, the *cost* of a smoothed score y with respect to x is defined as

(4) $c(y) = \gamma \cdot |\hat{y}| + d(x, y)$,

where γ is a penalty term that captures the cost of each new smoothed score and $d(\cdot,\cdot)$ is some distance function. It is also possible to have a node-specific penalty term γ_i for node i; for simplicity of exposition, we state the algorithm in terms of a node-independent term γ.

This cost function and the tree structure lead to a regularized version of the isotonic regression problem.

PROBLEM 2 (REGULARIZED TREE ISOTONIC REGRESSION). *Given a tree T and $x : T \to [0,1]$, find $y : T \to [0,1]$ that satisfies (1) and minimizes $c(y)$ as given by (4).* [1]

For the rest of the paper, we take $d(\cdot,\cdot)$ to be the L_1 norm since it is robust against outliers.

Before presenting the algorithm we discuss a key property of the L_1 distance measure that aids us in designing an efficient algorithm for this problem. We show that the optimal smoothed scores in y can only come from the classifier scores in x.

LEMMA 3. *There exists an optimal solution, $\hat{y}$, where, for all $i \in T$, if $\hat{y}(i) \neq \perp$, then there is a $j \in T$ such that $\hat{y}(i) = x(j)$.*

PROOF. Consider the maximal connected subtree T' of nodes in T such that (1) $i \in T'$, and (2) for all $j \in T'$, $y(j) = \hat{y}(i)$. If $\hat{y}(i)$ is not the median of the set of scores $\{x(j) \mid j \in T'\}$, then we can push $\hat{y}(i)$ closer to the median by a small amount and decrease the cost of the solution given by (4); this follows since the median is the minimizer for L_1 distance. □

[1]Note that a special case of our problem has been considered before in statistics and computer science contexts; it is usually referred to as the *isotonic regression problem*: given $\vec{x} = x_1, \ldots, x_n$, find $\vec{y} = (y_1, \ldots, y_n)$ such that $y_1 \leq \cdots \leq y_n$ and $d(\vec{x}, \vec{y})$ is minimized, where $d(\cdot,\cdot)$ is some distance function. It is easy to extend this definition to the case when the y_i's have to respect a given partial order, say, imposed by a tree.

We build a dynamic program using the above result to obtain an algorithm for the regularized tree isotonic regression problem. Algorithm BUILDERROR builds up an index function $\text{val}(i,j)$ and an error function $\text{err}(i,j)$ for each node $i \in T$. The value $\text{err}(i,j)$ represents the cost of the optimal smoothed scores in the subtree rooted at i if its parent node has the smoothed score $y(\text{parent}(i)) = x(j)$. In this situation, the index $\text{val}(i,j)$ is such that the optimal smoothed score for node i is given by $y(i) = x(\text{val}(i,j))$.

If $\text{val}(i,j)$ is the same as j, i.e, the optimal value for i and $\text{parent}(i)$ are the same $x(j)$, then the only cost is the L_1 distance between $x(i)$ and $x(\text{val}(i,j))$, otherwise there is an additional γ cost as well. The algorithm computes this error function by first computing errors as if the additional γ cost must always be added; this intermediate result is stored in the err' array, where $\text{err}'(j)$ is the error if the node under consideration has the smoothed score $y(i) = x(j)$. Then, it chooses between (a) continuing with the parent's value and subtracting γ from the corresponding cost err', or (b) creating a new section with a new value and paying in full the corresponding cost in err'. Once all error functions have been computed, the optimal smoothed scores are obtained using Algorithm ISOTONESMOOTH, which starts with the best index $p(\text{root}(T))$ at the root, and progressively finds the best index $p(\cdot)$ for nodes lower down in the tree.

Algorithm BUILDERROR (i, x, γ)

if (i is a leaf) then

1. for $j \in T$ /* all values node i can take */
   if $(x(i) - x(j) > \gamma)$ then
   $\text{err}(i,j) = \gamma; \text{val}(i,j) = i$
   else
   $\text{err}(i,j) = |x(i) - x(j)|; \text{val}(i,j) = j$

else

2. for child u of node i
   BUILDERROR(u, x, γ)
3. for $j \in T$ /* all values node i can take */
   $\text{err}'(j) = |x(i) - x(j)| + \sum_{k \in \text{child}(i)} \text{err}(k,j) + \gamma$
4. for $j \in T$ /* all values node $\text{parent}(i)$ can take */
   $\text{val}^* = \text{argmin}_{k \in T, x(k) > x(j)} \text{err}'(k)$
   $\text{err}^* = \text{err}'(\text{val}^*)$
   if $(\text{err}'(j) - \gamma \geq \text{err}^*$ or $i = \text{root}(T))$ then
   $\text{err}(i,j) = \text{err}^*; \text{val}(i,j) = \text{val}^*$
   else
   $\text{err}(i,j) = \text{err}'(j) - \gamma; \text{val}(i,j) = j$

Algorithm ISOTONESMOOTH (err, val)

$\text{val}^* = \text{argmin}_{i \in T} \ \text{err}(\text{root}(T), i)$
$p(\text{root}(T)) = \text{val}^*; y(\text{root}(T)) = x(\text{val}^*)$
for i in a breadth-first search order of T
$p(i) = \text{val}(i, p(\text{parent}(i))); y(i) = x(p(i))$

To demonstrate the correctness of this algorithm, we show that the restriction of the optimal solution to a subtree is also the optimal solution for the subtree under the monotonicity constraint imposed by its parent.

Consider the subtree rooted at any non-root node $i \in T$. Now suppose the smoothed score $y(\text{parent}(i))$ is specified. Then, let $z(\cdot)$ be the smoothed scores of the optimal solution to the regularized tree isotonic regression problem for this subtree, under the additional constraint that $z(i) \geq y(\text{parent}(i))$.

LEMMA 4. *For all nodes j in the subtree of i, $y(j) = z(j)$.*

PROOF. Consider a smoothed solution $w(\cdot)$ where $w(j) = z(j)$ for all nodes j in the subtree of i, and $w(j) = y(j)$ otherwise. Since $z(\cdot)$ obeys the monotonicity property and $z(i) \geq y(\text{parent}(i))$, the solution $w(\cdot)$ obeys the monotonicity property. Now, the cost $c(w)$ is the sum of the cost for the smoothed scores $z(j)$ in the subtree of i and the cost for the scores $y(k)$ for all other nodes. Thus, the difference between $c(w)$ and $c(y)$ is just the difference in costs for $z(j)$ and $y(j)$ in the subtree of i, for which we know that $z(\cdot)$ is the optimal. The lemma follows. □

THEOREM 5. *Algorithm* IsotoneSmooth *solves the regularized tree isotonic regression problem.*

PROOF. The algorithm computes up the optimal smoothed scores for each subtree, i.e., the $\text{err}(\cdot, \cdot)$ arrays, while maintaining (1) for every possible smoothed score of the parent. By Lemma 3, the parent can take only finitely many smoothed scores in the optimal solution, and by Lemma 4, combining the optimal smoothed scores for subtrees yields the optimal smoothed scores for the entire tree. □

Complexity. Let $|T| = n$. The space required per node is $O(n)$, and so the total space required is $O(n^2)$. Next, we consider the running time of the algorithm. In the dynamic program, step 1 takes $O(n^2)$ time, step 3 takes $O(n^2)$ time amortized over all calls, and step 4 can be done in $O(n^2 \log n)$ time by storing err' values in a heap and then running over the nodes $j \in T$ in ascending order of $x(j)$. Hence, the total running time is $O(n^2 \log n)$. This matches the time complexity of previously known algorithms for the non-regularized forms of tree isotonic regression [1].

5. DETAILS OF THE SYSTEM

In this section we describe the details of both the classification and smoothing aspects of our system.

5.1 Constructing training data

As mentioned before, we used a site-level template detection algorithm to generate training data for our classifier. The construction of training data involved two distinct steps: collecting webpages and obtaining labeled templates. We sampled 3,700 websites from the Yahoo! search engine index such that each website had at least 100 webpages. We also biased the sampling process slightly towards picking good quality host domains, and avoided picking pornographic or spam websites. Then, for each website we downloaded at most 200 randomly picked webpages.

All DOM nodes that occurred on more than 10% of the pages of any website were tagged as site-level templates. Since we wanted to learn a classifier for all internal DOM nodes we wanted representative labeled data from all levels of the DOM trees. Hence, for each internal node we computed how much of its HTML was part of a site-level template. DOM nodes with more than 85% of their HTML content within site-level templates were also labeled as templates. The rest of the DOM nodes were used as instances of the non-templates class.

Note that the condition required for tagging a node as template is very strong. This is done intentionally for two reasons. First, recall that a node is a non-template if *any* node in its subtree is a non-template. And second, the cost of misclassifying a non-template as a template is much higher than that of the reverse error.

5.2 Learning the classifier

There are multiple steps involved in learning the classifier. Each of these steps is described below.

Preprocessing. Each webpage is preprocessed and parsed so that features can be extracted for its DOM nodes. The preprocessing step involves cleaning the HTML code using Hypar[2], annotating the DOM nodes with position and area information using Mozilla[3], and parsing the HTML to obtain a DOM tree structure. The text in the HTML page is also processed to remove stop words.

Feature extraction. The training data that we employ for learning corresponds to site-level templates. However, we want our classifier to generalize to the global definition of templates. This makes the process of feature extraction very critical. From each DOM node, we extract features that we believe are indicative of whether or not that DOM node is a template. For example, intuitively, if the text within a DOM node shares a lot of words with the title of the webpage, then perhaps it is not a template node. Similarly, the distance of a DOM node from the center for the page indicates its importance to the main purpose of the page, and hence its templateness.

In a similar fashion, we constructed several other features from the position and area annotations of DOM nodes as well as from the text, links, anchortext they contain. The most discriminative features turned our to be: closeness to the margins of the webpage, number of links per word, fraction of text within anchors, the size of the anchors, fraction of links that are intra-site, and the ratio of visible characters to HTML content.

Classifier training. We trained Logistic regression classifiers [18] over the set of features described above. Apart from performing very well, these classifiers have the additional benefit that their classification output can be interpreted as the probability of belonging to the predicted class. In our exploratory experiments we observed that distributions of feature values varied heavily depending on the area of the DOM node. This is because template and non-template nodes have very different characteristics at different levels of the DOM trees; these levels can be approximated by the area of the node. Hence, we trained four logistic regression models for DOM nodes of different sizes. Now, given a webpage, the appropriate logistic model is applied to each node of the DOM tree, and the output probabilities are fed to our post-classification smoothing function.

5.3 Smoothing classifier scores

The smoothing algorithm allows arbitrary choices of penalty values for each tree node. However, in the domain of template detection, there are several desiderata that a good penalty function must try to achieve. We list these below, along with the particular functions that we considered, and the one that we finally settled upon.

Desiderata for penalties. There are three main desiderata for a smoothing algorithm in the context of template

[2] `www.cse.iitb.ac.in/~soumen/download`
[3] `www.mozilla.org`

detection. First, nodes that are too small in area should not form segments of their own. Such nodes have very little content, and their classification scores are unreliable. Also, having such small segments impairs the applicability of webpage segmentation to page visualization and browsing.

Second, adding nodes as segments should be easier as we move up from leaves to the root. The smoothed values assigned to nodes high up in the tree impose constraints on the possible values in their entire subtree. If creation of new segments is hard, such nodes may merge with other nodes to form larger segments whose smoothed scores may be drawn too far away from the classification score of the node itself, thus hurting all nodes in their subtree.

Third, if a child node accounts for a large fraction of the area of its parent node, then it should be harder to set the child to a value different from that of its parent. This encourages the smoothing algorithm to form large sections without too much nesting, which agrees with our intuitions about how webpage segments are created.

Handling very small nodes. All nodes whose area is less than 2000 sq pixels are neither classified nor smoothed; they are "hidden," and their effect is rolled into their parent node. Thus, a node with k hidden children acts as if it were $(k+1)$ nodes, all with the same classification value. This reduces to multiplying the distance measure $d(\cdot,\cdot)$ with $(k+1)$, and the smoothing algorithm can handle it trivially. This heuristic goes some way in achieving the first desideratum.

Penalty functions. We experimented with several penalty functions, attempting to achieve the aforementioned desiderata. Starting with a user-defined constant c, several transformations for γ_i (penalty for node i) were tried:
(1) $\gamma_i = c \cdot N/N_i$, where N_i is the number of nodes in the subtree rooted at i, and N is the total number of nodes in the DOM tree. This penalty is high for nodes near the leaves and low for nodes near the root, satisfying the first and second desiderata.
(2) $\gamma_i = c \cdot A/A_i$, where A_i is the area of node i, and A the area of the whole HTML page. Again, this penalty satisfies the first and second desiderata.
(3) $\gamma_i = c \cdot A_{\text{parent}(i)}/A_i$, where $A_{\text{parent}(i)}$ is the area of the parent of node i. This tries to achieve the third desideratum.

We tried all combinations of these penalties, over the a large range of the constant c, and visually inspected the results of smoothing on a few webpages. We finally settled on setting $\gamma_i = 0.01 \cdot A/A_i$, which gave the best results. For the rest of this paper, unless specified otherwise, the penalty is always set to this function.

6. EXPERIMENTS

We now present an empirical evaluation of our system, called PageLevel. Using human-labeled data, we show in Section 6.1 that our approach is very effective in detecting the template sections of webpages. Then, in Sections 6.2 and 6.3 we show that applying template detection as a preprocessing step significantly improves accuracy on standard web mining tasks such as duplicate webpage detection and webpage classification. We also show that template removal using PageLevel provides more benefits than using the more expensive SiteLevel approach.

Dataset		PageLevel Basic	PageLevel Smooth
Common	Text	0.56	**0.60**
	AT	0.65	**0.71**
	Links	0.69	**0.73**
Random	Text	0.63	**0.66**
	AT	0.71	**0.73**
	Links	0.75	**0.77**

Table 1: Accuracy of PageLevel on Common and Random datasets in terms of f-measure.

6.1 Template detection performance

The desiderata for a template detection system are as follows: (a) it must divide the webpage into segments separating template and non-template content; and (b) it must accurately identify the webpage segments as template or non-template. In this section we show that our system, PageLevel, achieves both these objectives.

Datasets. In order to evaluate the template detection performance of PageLevel we manually created two labeled datasets.

Common. We selected and manually labeled 44 pages from websites that were commonly visited by the authors. The selected pages come from a diverse set of domains, such as, news websites like NYTimes and CNN, university websites like UTexas-Austin, etc. For each webpage, the manual labeling process identified the largest possible HTML fragments that were either entirely template or non-template. These HTML fragments correspond to nodes in the webpage DOM tree. Hence, for each webpage we labeled an antichain of nodes through the DOM tree, forming an exhaustive and disjoint cover of all leaf nodes.

Random. In order to evaluate the algorithms on webpages more representative of the general Web, we manually labeled 100 pages selected uniformly at random from the DMOZ directory[4]. The selected set of webpages is a mix of topically focused content or hub pages and entry points to larger websites. As was done for the Common dataset, the labeling process identified for each webpage an antichain of DOM nodes and marked each node as either template or non-template.

Template detection accuracy. As we demonstrate later in this section, text and links present within the template regions mislead standard web-mining algorithms for tasks such as duplicate detection and automated classification. Here we measure the efficacy of PageLevel in identifying text and links that occur within templates. We report accuracy in terms of f-measure, which is the harmonic mean of precision (p) and recall (r); i.e. $f = 2pr/(p+r)$. In the current setting, precision is the fraction of words (links) identified by PageLevel as occurring within templates that are also manually placed within templates. Recall is the fraction of all words (links) manually labeled as lying within template regions that PageLevel also correctly identifies as templates. This evaluation setting has previously been used by Vieira et al. [24].

In Table 1, we present the accuracy numbers (in terms of f-measure) achieved by PageLevel for the two datasets:

[4] www.dmoz.org

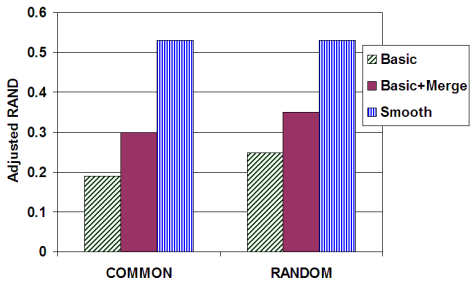

Figure 1: Segmentation performance of PageLevel Basic, PageLevel Basic+Merge, and PageLevel Smooth.

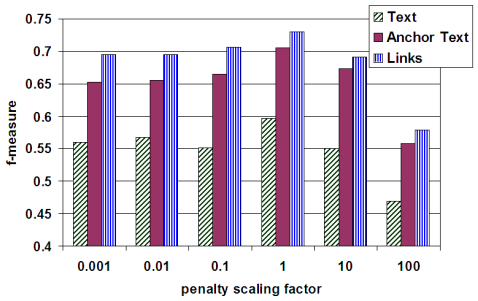

Figure 2: Variation in template detection accuracy on the Common dataset with changing values of penalty. The x-axis represents the factor being multiplied into the penalty.

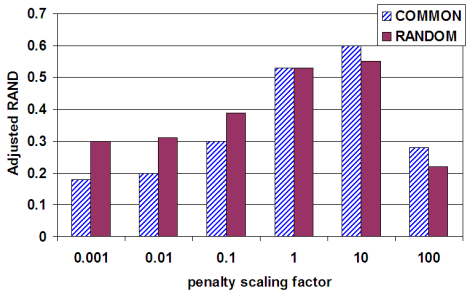

Figure 3: Variation in segmentation accuracy both datasets with changing values of penalty. The x-axis represents the factor being multiplied into the penalty.

Common and Random. We present accuracies for two variations of our approach: PageLevel Basic only applies the classifier to the DOM nodes individually, while PageLevel Smooth, in addition also performs Isotonic smoothing on the templateness scores. The performance is measured along multiple dimensions: Text, Anchor-text (AT), and Links. As is clear from the table, our approach is very effective in identifying all types of page content that lies within template regions. Furthermore, smoothing is shown to significantly improve accuracy across all dimensions of evaluation, in some cases by almost as much as 10%. An interesting observation is that the accuracy over the Common dataset is slightly lower than that over Random. This is because the template structure in webpages in Common is more extensive than those in Random. Further, note that the gains afforded by the isotonic smoothing are larger on the more difficult of the two datasets.

Segmentation accuracy. As we mentioned in Section 4, a by-product of the isotonic smoothing algorithm is a segmentation of the page into DOM nodes that act as the roots of the template and non-templates regions. Here, we show that the segmentation found by PageLevel closely matches the manually labeled segments.

Notice that the manual segmentation, an antichain of nodes of the DOM tree, induces a grouping of the leaves in which each node in the segmentation defines a group. A leaf then belong to the group corresponding to the segment node that covers it. Similarly, the segmentation output by PageLevel, even though it allows for nested segments, also induces a grouping of leaves. Each leaf can be considered as belonging to the group corresponding to its closest ancestor in the segmentation. Hence, we can evaluate the PageLevel segmentation against the manually labeled one by comparing the corresponding groupings using the adjusted RAND index [12]. The adjusted RAND index is a measure of how similar two groupings are, i.e., whether pairs of objects (leaves) are together in both groupings, or in different groups in both groupings. It is used as a preferred measure of agreement between clusterings [17]. The value of the adjusted RAND index is upper bounded by 1, and its expected value for a random clustering is 0.

In Figure 1 we plot the accuracy of PageLevel segmentation in terms of adjusted RAND. The PageLevel Basic and PageLevel Smooth approaches have been described above. As we can see the PageLevel Basic algorithm achieves close to random results, but this is expected since it is not performing any smoothing of scores and hence almost every leaf is in a group of its own. In contrast the segmentation discovered by the isotonic smoothing function (PageLevel Smooth) conforms very well to the manually labeled segments. In order to put the accuracy of PageLevel Smooth in context, we also present numbers for a PageLevel Basic + Merge heuristic. This approach does a "naive" smoothing of the classifier scores by grouping adjacent leaves together when their templateness scores differ by less than δ (the best δ was found by exhaustive search). As we can see from the plot, "merging" improves the scores of the PageLevel Basic; however, the results are still far lower than those achieved by isotonic smoothing. This shows that the smoothing operation is constructing highly non-trivial segmentations of webpages.

Effect of variations in penalty. We have shown above that PageLevel successfully obtains and labels template segments within webpages. Here we discuss the sensitivity of our approach to penalty parameters in the isotonic smoothing function.

Figure 2 plots the variation in template detection accuracy on the Common dataset with changing values of penalty. In the plot, the x-axis represents the factor multiplied into the penalty in order to vary it. As we can see, an increase or decrease in penalty results in an decrease in the template detection accuracy. However, the decrease is larger with higher values of penalty as this results in very few segments and hence a mixing up of template and non-template struc-

	Total Pairs	PageLevel	SiteLevel	FullText
Dup	1711	**1299** (**76**%)	730 (42.7%)	529 (30.9%)
Non-Dup	2058	**1885** (**91.6**%)	1712 (83.2%)	1781 (86.5%)

Table 2: Number of duplicate and non-duplicate pairs detected by the shingling approach after removing templates detected by PageLevel and SiteLevel . FullText indicates no template detection and removal.

tures into the same segment. Lower values of penalty do not give us the improvements inherent in smoothing, but they do not reduce the discriminative power of the Basic classifier. The same behavior is seen for Random as well.

Variations in segmentation accuracy on both datasets with changing values of penalty are plotted in Figure 3. Just as in the case of template detection accuracy, the segmentation accuracy also forms a unimodal curve, dropping with high and low values of penalty. However, the drop in segmentation accuracy is larger for changes in penalty values as compared to drop in detection accuracy. This is because the smoothing impacts the segmentation performance more directly, as compared to detection performance. As we increase (decrease) penalty values the number of groups of leaves obtained are lesser (greater) than the manually labeled groups. Both these changes negatively impact the segmentation performance. Another interesting difference is that template detection accuracy achieves high values even when segmentation performance is not at its peak. The reason is that the manual labeling is binary (template or non-template), while the segments we find are labeled with real numbered scores. Hence, we can still achieve a high template detection accuracy when the smoothing function places leaves into groups smaller than those in the manual labellings. However, groups smaller than those in the manual labeling causes a decrease in segmentation accuracy. This indicates that if achieving good segmentation is our primary objective, using a slightly higher value of penalty might be advantageous.

To summarize, in this section we showed that PageLevel accurately segments webpages, and also labels the segments appropriately as template or non-template. Further, we showed that isotonic smoothing is critical to its success, contributing to increases in both segmentation and template detection accuracies. We were unable to provide any comparisons with the site-level approach on the human labeled data, since SiteLevel needs many pages from each website in order to make template judgments for pages.

Next we show that webpage template detection is very useful as a pre-processing step in several applications, such as finding webpages with duplicate content, and webpage classification. Furthermore, since in these datasets we have several webpages from the same website available, we also present an evaluation comparing PageLevel with the site-level template detection approach.

6.2 Application to duplicate detection

Duplicate webpages and mirrored websites present challenging problems to web search engines that crawl and index them. Duplicated pages use up valuable index space and duplicate results returned for search queries spoil the user experience. Hence, detection of duplicates on the Web in a scalable fashion has been the subject of much research [4, 5, 8]. Most duplicate detection methods rely on the concept of shingles. For each webpage, shingles are extracted by moving a window of fixed length over the text, and the ones with the N smallest hash values are stored. Two documents that share shingles are then considered to be near-duplicates.

Problems caused by templates. The templates regions often contain text whose purpose is orthogonal to the main content of the webpage. Hence, this templated content must not be used while making decisions about whether pages are duplicates. For example, text present within navigation bars, copyright notices etc., must not be compared when two pages are being checked for duplicate material. The presence of templated content of webpages can foil duplicate detection algorithms whenever the shingling process retains shingles from the templated regions. Two pages that have absolutely the same content, say the exact same AP news story repeated across two different news websites, might be considered non-duplicates if the shingling process retains shingles from the template regions of the webpages as this portion of the webpages is different. This can lead to false negatives and cause us to return duplicate results for queries. Similarly, two webpages with the same templated content but different main content might be considered duplicates if all the shingles hit the templated region. This can result in false positives and cause us to ignore valuable content on the web. In this section we evaluate the effect of templates on duplicate detection performance, and also also compare the template detection performance of PageLevel to the site-level approach.

The Lyrics dataset. We constructed the Lyrics dataset by obtaining the webpages containing lyrics for the same song from three different websites. This way we knew that the webpages from different websites containing lyrics to the same song should be considered duplicates[5]. We also knew that webpages containing lyrics of different songs, irrespective of what website they come from, should be considered non-duplicates. We were able to obtain 2359 webpages from the websites `www.absolutelyrics.com`, `www.lyricsondemand.com`, and `www.seeklyrics.com` containing lyrics to songs by artists ABBA, BeeGees, Beatles, Rolling Stones, Madonna, and Bon Jovi. We chose to get lyrics by a few diverse artists in order to minimize the possibility of cover songs. The Lyrics dataset consists of 1711 duplicate pairs (webpages with lyrics of the same song from different websites) and 2058 non-duplicate pairs (webpages with lyrics of different songs from the same website).

Experimental setup. SiteLevel was run on all the pages on each lyrics website and the threshold parameter was set to 10%. This setting was seen to perform well in [11].

We used a standard shingling process. Before the shingling was performed the text of the webpage is made lowercase and only alphanumeric characters are retained. Shingles are computed over moving windows of 6 consecutive words each, and the 8 minimum hashes are stored for each webpage. A pair of pages is tagged as a duplicate if there are at least 4 matching hashes out of the 8 for each webpage.

[5]Actually, these might only be near-duplicates, due to transcription errors on the different pages. However, this affects all algorithms equally.

Categories		FullText	SiteLevel	PageLevel Basic	PageLevel Smooth
camera	mobile	55.10	**64.17**	59.57	60.17
camera	notebook	30.48	35.61	35.24	**40.03**
camera	printer	32.76	38.84	39.75	**41.18**
camera	tv	34.82	**40.67**	39.51	37.21
mobile	notebook	60.45	**70.26**	64.94	66.92
mobile	printer	21.06	24.16	24.5	**28.55**
mobile	tv	23.03	23.86	**24.85**	24.79
notebook	printer	39.9	43.95	48.7	**53.91**
notebook	tv	43.47	48.85	50.2	**50.94**
printer	tv	41.17	44.12	**48.7**	47.57
Average		38.22	43.45	43.6	**45.13**

Table 3: Averaged classification accuracies on 2-class problems. The best accuracies for each class combination are in bold.

We run this experiment under different settings: (1) all segments (and words) are used (FullText), (2) only segments tagged as non-template by PageLevel are used, and (3) only segments tagged as non-template by SiteLevel are used for shingling.

Results. The results of our duplicate detection experiments are presented in Table 2. Not detecting and removing template content (FullText) performs very badly, especially in flagging duplicate pairs. Using the PageLevel approach to clean the data before shingling recovers 76% of the duplicate pairs, and 92% of non-duplicate ones. These represent an improvement of 140% and 6% respectively over the FullText approach. Finally, PageLevel also outperforms the SiteLevel template detection approach by a large margin in both flagging duplicate and non-duplicate pairs.

The Lyrics dataset is not representative of the density of duplicates and non-duplicate pairs found on the web; we created it to highlight the problems posed by templates to duplicate detection algorithms. Hence, while the numbers seen in these experiments will not apply exactly to the web in general, the results are indicative of the benefits of template detection and removal, and the dataset serves as an appropriate test-bed for comparing the algorithms PageLevel and SiteLevel.

Discussion. Why does PageLevel outperform SiteLevel even though it is trained on the output of the latter? Comparing errors performed by the two on the LYRICS dataset offers us an opportunity to investigate this. As stated before, errors occur when shingles come from templated regions of the page. Many of the errors committed by SiteLevel involved shingles from a segment on "Popular lyrics by this Artist" that seemed to change based on the artist whose song lyrics were being displayed. Since this segment changed within webpages on the same site, SiteLevel was unable to identify it as a template. However, thanks to the careful selection of DOM node features in the page-level classifier, PageLevel generalizes beyond the site-level training data. Thus, it picked out such segments as templates, boosting its accuracy significantly.

6.3 Application to webpage classification

Automated classification of webpages is a well-studied problem and numerous approaches have been proposed for it. While many sources of information like hyperlinks [6], site structure [15], etc. are often used, techniques for classifying the text content of webpages [18] form the mainstay of webpage classification. In this section we perform experiments on the effect of template content on the classification of textual content of webpages, and show that template removal using PageLevel gives a boost in accuracy.

Problems caused by templates. Even though classification algorithms are very good at identifying and removing noisy features, in certain scenarios templates can present a challenging problem. Consider a binary classification problem between classes *Camera* and *Notebook*. If the template terms (noise) in both classes are the same, a classifier would be able to detect and remove it, say, using the correlation of features to the class labels. However, the noisy features could differ across the two classes; say, the webpages in *Camera* class come from CNET, and those in *Notebook* class come from PCConnection, the classifier will not be able to remove the template content automatically, making template detection as a pre-processing step imperative.

Dataset and experimental setup. For the classification experiments we used a subset of the dataset used by Vieira et al. [24]. The dataset consisted of webpages on 5 topics (*Camera, Notebook, Mobile, Printer, TV*) obtained from 4 websites, CNET, J&R, PCConnection, and ZDNET. Details of this dataset can be obtained in [24]. From this data, we constructed binary classification problems in which training data for classes C_1 and C_2 were taken from different websites, and the rest of the data for these classes were used as test data. For instance, in one binary problem, C_1 is *Camera* and C_2 is *Notebook*. The training data for C_1 and C_2 comes from CNET and PCConnection respectively. The test data for C_1 then comprises J&R, PCConnection, and ZDNET, while that for C_2 comprises CNET, J&R, and ZDNET. This evaluation setting has been used previously [24, 26]. We employed a Naive Bayes classifier[6] for the binary classification problems. The classification accuracy numbers we report are averaged over all possible binary classification problems of the type mentioned above.

We run this experimental setup with different amounts of webpage cleaning: (1) with all segments (and words) (FullText), (2) with only segments tagged as non-template by SiteLevel, (3) only segments tagged as non-template by the PageLevel Basic algorithm (4) only segments tagged as non-template by the PageLevel Smooth algorithm. Recall

[6] www.cs.cmu.edu/~mccallum/bow/

that in PageLevel Basic algorithm the smoothing function is disabled, and only raw templateness classifier assigned scores are used.

Results. The results of the classification experiments are presented in Table 3. The best accuracies for each class combination are highlighted in bold. As we can see, using template detection as a pre-processing step always improves the classification accuracy of the Naive Bayes classifier. Furthermore, webpage cleaning via the PageLevel algorithm outperforms SiteLevel on a majority of class combinations. Even among the PageLevel approaches, the use of isotonic smoothing of the templateness classifier's output results in better template detection and removal, as evidenced by an increase in the Naive Bayes classifier accuracy. In the final analysis, webpage template detection and removal via our PageLevel system increases classification accuracy on this dataset by an average of 18%.

7. CONCLUSIONS

We presented a framework for classifier based page-level template detection that constructs the training data and learns the notion of "templateness" automatically using the site-level template detection approach. We formulated the smoothing of classifier assigned templateness scores as a regularized isotonic regression problem on trees, and presented an efficient algorithm to solve it exactly; this may be of independent interest. Using human-labeled data we empirically validated our system's performance, and showed that template detection at the page-level, when used as a preprocessing step to webmining applications, such as duplicate detection and webpage classification, can boost accuracy significantly.

Acknowledgments. We thank Rajat Ahuja, Rupesh Mehta, Arun Ramanujapuram, and Amit Sasturkar for their valuable help. We also thank the reviewers for their helpful comments.

8. REFERENCES

[1] S. Angelov, B. Harb, S. Kannan, and L.-S. Wang. Weighted isotonic regression under the l_1 norm. In *Proc. 17th SODA*, pages 783–791, 2006.

[2] S. Baluja. Browsing on small screens: Recasting web-page segmentation into an efficient machine learning framework. In *Proc. 15th WWW*, pages 33–42, 2006.

[3] Z. Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. In *Proc. 11th WWW*, pages 580–591, 2002.

[4] K. Bharat, A. Broder, J. Dean, and M. R. Henzinger. A comparison of techniques to find mirrored hosts on the WWW. *JASIS*, 51(12):1114–1122, 2000.

[5] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *WWW6 / Computer Networks*, 29(8-13):1157–1166, 1997.

[6] S. Chakrabarti, B. E. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *Proc. SIGMOD*, pages 307–318, 1998.

[7] Y. Chen, X. Xie, W.-Y. Ma, and H.-J. Zhang. Adapting web pages for small-screen devices. *Internet Computing*, 9(1):50–56, 2005.

[8] J. Cho, N. Shivakumar, and H. Garcia-Molina. Finding replicated web collections. In *Proc. SIGMOD*, pages 355–366, 2000.

[9] B. Davison. Recognizing nepotistic links on the web. In *AAAI-2000 Workshop on Artificial Intelligence for Web Search*, pages 23–28, 2000.

[10] S. Debnath, P. Mitra, N. Pal, and C. L. Giles. Automatic identification of informative sections of web pages. *TKDE*, 17(9):1233–1246, 2005.

[11] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of web page templates. In *Proc. 14th WWW (Special interest tracks and posters)*, pages 830–839, 2005.

[12] L. Hubert and P. Arabie. Comparing partitions. *J. Classification*, 2:193–218, 1985.

[13] H.-Y. Kao, M.-S. Chen, S.-H. Lin, and J.-M. Ho. Entropy-based link analysis for mining web informative structures. In *Proc. 11th CIKM*, pages 574–581, 2002.

[14] H.-Y. Kao, J.-M. Ho, and M.-S. Chen. WISDOM: Web intrapage informative structure mining based on document object model. *TKDE*, 17(5):614–627, 2005.

[15] R. Kumar, K. Punera, and A. Tomkins. Hierarchical topic segmentation of websites. In *Proc. 12th KDD*, pages 257–266, 2006.

[16] N. Kushmerick. Learning to remove internet advertisement. In *Proc. 3rd Agents*, pages 175–181, 1999.

[17] G. Milligan and M. Cooper. A study of the comparability of external criteria for hierarchical cluster analysis. *Multivariate Behavioral Research*, 21(4):441–458, 1986.

[18] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[19] T. Morton-Jones, P. Diggle, L. Parker, H. O. Dickinson, and K. Blinks. Additive isotonic regression models in epidemiology. *Statistics in Medicine*, 19(6):849–859, 2000.

[20] P. M. Pardalos and G. Xue. Algorithms for a class of isotonic regression problems. *Algorithmica*, 23(3):211–222, 1999.

[21] T. Robertson, F. T. Wright, and R. L. Dykstra. *Order-Restrictied Statistical Inference*. Wiley, 1988.

[22] R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma. Learning block importance models for web pages. In *Proc. 13th WWW*, pages 203–211, 2004.

[23] Q. Stout. Optimal algorithms for unimodal regression. *Computing Science and Statistics*, 32:348–355, 2000.

[24] K. Vieira, A. Silva, N. Pinto, E. Moura, J. Cavalcanti, and J. Freire. A fast and robust method for web page template detection and removal. In *Proc. 15th CIKM*, pages 256–267, 2006.

[25] L. Yi and B. Liu. Web page cleaning for web mining through feature weighting. In *Proc. 18th IJCAI*, pages 43–50, 2003.

[26] L. Yi, B. Liu, and X. Li. Eliminating noisy information in web pages for data mining. In *Proc. 9th KDD*, pages 296–305, 2003.

[27] X. Yin and W. S. Lee. Using link analysis to improve layout on mobile devices. In *Proc. 13th WWW*, pages 338–344, 2004.