World Wide Web
DOI 10.1007/s11280-013-0269-6

Entropy-based automated wrapper generation for weblog data extraction

George Gkotsis · Karen Stepanyan · Alexandra I. Cristea · Mike Joy

Received: 31 October 2012 / Revised: 24 October 2013 / Accepted: 4 November 2013

Abstract This paper proposes a fully automated information extraction methodology for weblogs. The methodology integrates a set of relevant approaches based on the use of web feeds and processing of HTML for the extraction of weblog properties. The approach includes a model for generating a wrapper that exploits web feeds for deriving a set of extraction rules automatically. Instead of performing a pairwise comparison between posts, the model matches the values of the web feeds against their corresponding HTML elements retrieved from multiple weblog posts. It adopts a probabilistic approach for deriving a set of rules and automating the process of wrapper generation. An evaluation of the model is conducted on a collection of weblogs reporting a prediction accuracy of 89 %. The results of this evaluation show that the proposed technique enables robust extraction of weblog properties and can be applied across the blogosphere.

Keywords Web information extraction · Automatic wrapper generation · Weblogs

This work was conducted as part of the BlogForever project funded by the European Commission Framework Programme 7 (FP7), grant agreement No.269963.

G. Gkotsis (✉) · K. Stepanyan · A. I. Cristea · M. Joy
Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK
e-mail: g.gkotsis@warwick.ac.uk

K. Stepanyan
e-mail: K.Stepanyan@warwick.ac.uk

A. I. Cristea
e-mail: A.I.Cristea@warwick.ac.uk

M. Joy
e-mail: M.S.Joy@warwick.ac.uk

Published online: 21 November 2013

1 Introduction

The problem of web information extraction dates back from the early days of the web and is fascinating and genuinely hard. The web, and the Blogosphere as a constituent part, correspond to a massive, publicly accessible data source. However, the scale is not the only challenge for capturing web resources. The heterogeneous nature of these resources, the large numbers of third party elements and advertisements, the rapid changes, the propagation of user-generated content and the diversity of inter-relations across the resources are among the common characteristics of the web. These characteristics amplify the complexity of capturing, processing and transforming of these web resources into structured data. The successful extraction of weblog properties and its transformation into structured data is of paramount importance for improving the quality of analysis, searching and preservation of such resources.

Although the exact number of weblogs is not known, it is evident that the size of the blogosphere is large and continues to grow. According to a survey published in October 2013 by W3Techs [25], WordPress, one of the most popular blogging engines, is used by 20.3 % of all the websites, while having a market share of 58.8 % amongst content management systems. In 2008 alone, the Technorati reported to be tracking more than 112 million weblogs, with around 900 thousand blog posts added every 24 hours [27]. The dataset of updated weblogs [5], as captured by Spinn3r between 13.01-14.02.2011, contains 133 million (mainly English) blog posts. In Britain, 25 % of Internet users maintain weblogs or personal websites [7] that are read by an estimated 77 % of Web users [27]. Hence, the volume of information published on weblogs justifies the attention of information retrieval, preservation and socio-historical research communities.

Unlike static web pages, weblogs are commonly perceived as dynamic and versatile web spaces. They often promote social connectedness, sharing, content creation and collaboration. Weblogs vary in their choice of subject, writing style, purpose, media use, platform and presentation. They can represent individuals, organisations or companies and connect to different audiences. Weblogs can assemble into specialised communities and interweave into sparse networks. This investigation focuses on data extraction techniques suitable for weblogs. Prior to proceeding to the subject of data extraction, it is necessary to outline the *weblog* as the object of data extraction.

Typically, the content of a weblog resides in a relational database, though alternative solutions for managing and maintaining the data are becoming available.[1] At the time of accessing a weblog (that is, requesting the resource via the HTTP protocol), it is being dynamically compiled by the blogging platform and sent to the web browser for rendering. The automation supported by the blogging platform provides a common structure that can be observed across the various weblog pages. More specifically, the weblog post, which constitutes a building block of a weblog, is comprised of a set of properties, such as the title, author, publication date, post content and the categories (or tags) assigned. Whilst the data structure is presented in a consistent way across a certain weblog, it rarely appears identical across weblogs, even if the blogging platform remains the same. The main reason for the above inconsistency is the fact that bloggers are allowed to personalise the presentation of their weblog arbitrarily in numerous ways, hence the resulting weblog exhibits a customised and unique document structure.

[1] http://www.instantfundas.com/2009/09/blogging-without-database-7-database.html

The above settings are shaping a challenging problem for state-of-the-art wrappers, since the extraction rules cannot automatically adapt to each weblog case. Even in the case of unsupervised wrapper induction, where examples are not provided by the user and user feedback is not required during training, human effort is typically required. This effort concerns the selection of the initial web pages or the manual annotation of the output of information extraction, such as missing attributes and ambiguous data. In our case, in order to deploy a pragmatic weblog data extraction solution, information extraction must be fully automated and be able to deal with issues that usually require human intervention.

One of the most prominent characteristics of weblogs is the existence of web feeds. Web feeds, commonly provided as RSS, are XML documents that allow access to the content of a website, such as a weblog, through a machine interpretable, structured document. Due to their rigorous structure, web feeds are used for accessing the content of a weblog in several applications. For instance, ArchivePress[2] is a plugin built for WordPress,[3] which relies on and is limited by the weblogs' web feeds [22]. This limitation appears due to the fixed number of entries found in a web feed and a recent survey on web feeds reports that the typical number of posts found in a web feed is 10 [21]. Furthermore, the content of the posts found in the web feed might be clipped and may contain only a summary of the weblog post.

The solution proposed here enables to overcome the above limitations. Intuitively, the idea is not to treat the web feeds as the only source of information, but as a medium that will allow the training and generation of a wrapper automatically. During this unsupervised training session, the matching of the elements found between the web feeds and the web documents is used to observe and record the position and the properties of the cross-matched elements. Based on these observations, a set of rules is generated through an inherently probabilistic approach. The paper is structured as follows. Section 2 introduces the concept of wrapper and wrapper generation and Section 3 describes the model proposed. Section 4 addresses the problem of matching text values between different sources, a crucial problem upon which the proposed solution relies. Section 5 presents a simple, real-world example as a demonstration of wrapper generation. Finally, Section 6 evaluates the model, Section 7 discusses the contribution of the approach and Section 8 presents the conclusions.

2 Wrapper generation

Work on using wrappers for data extraction started a few years after the emergence of the web. Initially, the wrapper was a custom built tool developed to parse specific web pages. This page- or site-level approach required a lot of effort and was inflexible to website changes. Traditionally, a wrapper is considered a semi-automatic approach, due to the requirement for labelling a collection of pages manually prior to applying the rules for extracting target data from other similarly formatted pages [18]. However, developments in wrapper generation have yielded automatic, unsupervised approaches as well. Hence we adopt the general definition by Baumgartner et al., as follows:

> A wrapper is a program that identifies the desired data on target pages, extracts the data and transforms it into a structured format [3].

[2] http://archivepress.ulcc.ac.uk/

[3] http://wordpress.org/

In our approach, as stated in the definition above, the identification of target pages is not part of the wrapper. In general, this problem is typically resolved as part of the crawling process. We refer the reader to the discussion of the problem of identification of target pages and how our work can inform this process in Section 7.

From a more functional perspective, Laender et al. [17] describe a wrapper as a program that executes a mapping W, which populates a data repository R with the objects found in a webpage S. This mapping must also be capable of recognizing and extracting data from any other page S' similar to S. The development of a wrapper is called *wrapper generation* or *induction*. Developing a wrapper can take different forms that use wrapper generation languages or pattern matching. The term *wrapper induction* is often used to describe an inductive process of wrapper generation. It may, for instance, involve application of supervised learning. Therefore, the term *wrapper generation* is more general.

3 Proposed model

As already mentioned, our model enables to generate a fully automated wrapper. It can be classified as an unsupervised data extraction method for weblogs. The approach is divided into three steps as follows.

3.1 Step 1: Feed processing and capturing of post properties

The only prerequisite of the first step is the acquisition of the weblog's feed URL. During this step, the content of the feed is fetched and processed through an ordinary XML parsing library. A feed contains a section of entries that point to corresponding weblog posts. For each entry, the following attributes are typically found and stored.

- *Title*: An optional string. A post can contain no more than 1 title.
- *Author*: An optional string.
- *Date Published*: A required object of type date. A post contains exactly 1 publication date.
- *Summary*: An optional HTML encoded part of the weblog post's content (usually the beginning of the post's content).
- *Permalink*: A required URL that points to the actual weblog post.
- *Categories*: An optional collection of words/phrases describing the topic of the post.

3.2 Step 2: Generation of filters

The second step includes the generation of filters. The concept of a *filter* has already been used in research related to web information extraction. Baumgartner et al. [2] use the term to refer to the building block of patterns, which in turn describe a generalised tree path in the HTML parse tree. Adding a filter to a pattern extends the set of extracted targets, whereas imposing a condition on a filter restricts the set of targets. The same concept is used by XWRAP [19] in order to describe the so-called "declarative information extraction rules". These rules are described in XPath-like expressions and point to regions of the HTML document that contain data records.

Following related work, we use the concept of a filter in order to identify and describe specific data elements of an HTML weblog post. Unlike past approaches where most of the

tools deal with the absolute path only (for example through partial tree alignment [30]), our filters are comprised of triples, which inform and extend existing solutions. Our approach overcomes irregularities appearing across absolute path values by providing additional means of describing the HTML element (namely the CSS Classes and HTML Identifiers). Our evaluation will later show that especially CSS Classes are used as extraction rules successfully many times, a feature that remains unexploited in most (if not all) approaches until now. Furthermore, in our approach, we acknowledge that neither of these three features (attributes) alone is strong enough to serve as the means for robust extraction. Instead, we assert that the combination of these three features is a vigorous solution.

In our approach, the filter is described using three basic attributes: the Absolute Path, the CSS Classes and the identifiers of the HTML element:

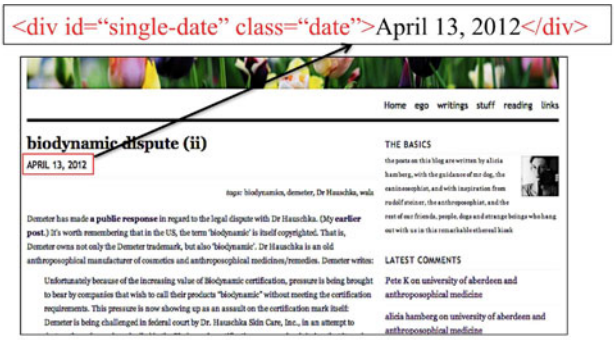
- Absolute path: XPath[4] (XML Path Language) is a W3C-defined language initially introduced in 1999 that is used to address parts of an XML document. There are roughly two basic approaches when addressing an element: the absolute and relative path. The Absolute Path is the path from the root of an HTML document to the desired element. This element may either be a node of the DOM tree or simply an end-node (leaf). Thus, the Absolute Path is described as a sequence of edges, where every edge is defined as the *name* of the element and the positional information of the element (*index*)[5] This sequence of edges starts from the root of the document and ends with the element containing the value we are interested in.
- CSS Classes: CSS (Cascading Style Sheets) are "a simple mechanism for adding style (e.g., fonts, colours, spacing) to web documents",[6] first introduced in 1996. It allows the separation of document content from its presentation through the definition of a set of rules. The rules contain information about the selectors and a declaration block.
- HTML Identifiers: The ID attribute identifies uniquely an HTML element of a document. It is commonly used in cases where CSS code needs to address one specific, unique element (e.g. the title of a post) or run JavaScript.

Figure 1 shows the structure of a filter with an annotated example. When pointing at a specific element, a set of HTML Identifier values and CSS Classes together with a single-valued Absolute Path are used to describe and define the filter. More specifically, when an element is identified, any HTML Identifiers or CSS Classes applied to this element are added to the filter. Afterwards, an iterative selection of the parent element continues, adding HTML Identifiers and CSS Classes to the corresponding sets, as long as the value of the parent element contains nothing but the value identified. For the example of Figure 1, the value for the ID attribute is *single-date*, for the CSS Classes the value is *date* and the Absolute Path is *html[0]/body[1]/div[1]/div[1]/div[0]/div[0]/div[1]*. Furthermore, the filters are generated as the result of matching the post properties against the actual HTML document; technical details about this matching process are provided in Section 4.

[4] http://www.w3.org/TR/xpath/

[5] The positional information of an HTML element is crucial in HTML documents, since this affects its visual representation. This is one of the reasons that HTML DOM trees are viewed as labelled ordered trees in the literature. (e.g., [10]).

[6] http://www.w3.org/Style/CSS/



Filter		
IDs	CSS Classes	Absolute Path
single-date	date	html[0]/body[1]/div[1]/div[1]/div[0]/div[0]/div[1]

Figure 1 The structure of a filter. An example is annotated for the case of an element containing a date value

3.3 Step 3: Induction of rules and blog data extraction

After the completion of step 2, a collection of filters is populated for each property. The values of these filters are such that, when applied to the weblog posts from which they were extracted, they link back to the HTML element containing the value of the matched property. However, due to multiple occurrences of values during the text matching process of Step 2, there are cases where a value is found in more than one HTML element. This results in generating a number of filters equal to the number of values found. Therefore, the collected filters are not yet suitable for extracting the data out of a weblog; the collection contains diverse and conflicting information that needs further processing. This "filter pollution" (or "diversity") may appear due to the following reasons.

- The visual theme of each weblog is different. This directly affects the values of the filter attributes. Hence, filters cannot be generally applied across different blogs.
- Even for a single weblog, the blogging platform may choose to render differently the property to be found across different posts. For example, as the size of the post content changes, the Absolute Path is affected. This value of the path, if reused elsewhere, may point to different or null elements.
- The value to be matched may be repeated more than once in a single web page. For instance, the title of a weblog post may be included at the sidebar under the section "recent posts" or the name of the author may be cited several times. Since our approach produces a filter for each matched value, multiple filters refer to and describe the same property.
- There are cases where a part of a filter should be applied in order to extract the desired property. For example, there are cases where the value of an Absolute Path (or an auto-incrementing HTML identifier) changes for each instance of a weblog post. At the same time, an identical CSS Class value across posts might identify the desired property

successfully. In that case, only the CSS Class value should be used to extract the desired property.

The goal of this step is to address the above issues. More specifically, the aim is to exploit the knowledge residing in the filters in order to generate a *set of rules* that describes how to extract the properties for a weblog automatically. These rules are devised from the collection of the filters already accumulated. They are described in filter attributes and direct the process of extraction.

In the case of weblog data extraction, there is neither prior knowledge of the location of the elements to be identified, nor a definite, automated way to describe them. Instead, we propose a case-based reasoning mechanism that assesses the information found in filters. The aim of this mechanism is to generate rules through a *learning by example* methodology, i.e., a general *rule* for each property is extracted through the examination of a set of *instances*. In our case, the *instances* correspond to the weblog posts that lead to the generation of the filters during the previous step. The *rules* are defined in the language used to describe the previously collected filters and we propose to compute them in an inherently probabilistic way, so as to accommodate irregularities found in web documents.

Moreover, the aim is to account for each attribute of each filter (Absolute Path, CSS Values and HTML identifiers) individually, in order to assess the utility and the likelihood of each attribute value as a rule. An important consideration here is the fact that selecting a "best-match" filter from the list of the filters or a non-empty value for *each* attribute (as a collection of "best-of" values for each attribute) may result in the elimination of the desired element. For instance, consider the case where an HTML identifier might increment for each weblog post and is therefore unique for every instance; in that case HTML identifiers should not be used at all.

Additionally, since the information gathered in the filters contains a degree of uncertainty, we propose to use the concept of *entropy* for solving the problem of devising the extraction rules. Information *entropy* and information *gain* have been used in classification problems, such as the ID3 algorithm [23]. Entropy, which was initially studied by Shannon, is defined as a measure of uncertainty or randomness of a phenomenon [13]. The entropy $E(S)$ of the set S for a given attribute is defined as:

$$E(S) = -\sum_{j=1}^{n} f_s(j) \log_2 f_s(j)$$

where:

- n is the number of different values of the attribute in S;
- $f_s(j)$ is the frequency of the value j in the set S.

Our approach adopts the concepts of entropy-based measurements [11] with the aim of generating a set of rules. In our case, instead of considering the entropy of each attribute, the entropy of the *value* of each attribute (called *property* in this paper and denoted as A_i) is accounted, which is:

$$E(S_{A_i}) = -f_s(A_i) \log_2 f_s(A_i)$$

Hence the gain (according to Quinlan [23]), which is now defined in relation to every value i of its attribute A (denoted as $G(S, A_i)$) over its set, is:

$$G(S, A_i) = -f_s(A_i) \cdot E(S_{A_i})$$

Table 1 A simple example of different filters. ∅ denotes that no HTML Identifier was found

HTML Identifiers	CSS Classes	Absolute Path
∅	cssValue1	path1
id1	cssValue1	path1
id1	cssValue1	path2
∅	cssValue2	path2
∅	cssValue2	path3

For example, consider the example in Table 1. For this table, we have:

$$
\begin{aligned}
E_{IDs_{null}} &= -\tfrac{1}{5}\log_2\left(\tfrac{1}{5}\right) = -0.46^{7} \\
G(S, IDs_{null}) &= -\tfrac{1}{5}(-0.46) = 0.09 \\
E_{IDs_{id1}} &= -\tfrac{2}{5}\log_2\left(\tfrac{2}{5}\right) = -0,53 \\
G(S, IDs_{id1}) &= -\tfrac{2}{5}(-0.53) = 0.21 \\
E_{CSS_{cssValue1}} &= -\tfrac{3}{5}\log_2\left(\tfrac{3}{5}\right) = -0.44 \\
G(S, CSS_{cssValue1}) &= -\tfrac{3}{5}(-0.44) = 0.27 \\
E_{CSS_{cssValue2}} &= -\tfrac{2}{5}\log_2\left(\tfrac{2}{5}\right) = -0.53 \\
G(S, CSS_{cssValue2}) &= -\tfrac{2}{5}(-0.53) = 0.21 \\
E_{Path_{path1}} &= -\tfrac{2}{5}\log_2\left(\tfrac{2}{5}\right) = -0.53 \\
G(S, Path_{path1}) &= -\tfrac{2}{5}(-0.53) = 0.21 \\
E_{Path_{path2}} &= -\tfrac{2}{5}\log_2\left(\tfrac{2}{5}\right) = -0.53 \\
G(S, Path_{path2}) &= -\tfrac{2}{5}(-0.53) = 0.21 \\
E_{Path_{path3}} &= -\tfrac{1}{5}\log_2\left(\tfrac{1}{5}\right) = -0.46 \\
G(S, Path_{path3}) &= -\tfrac{1}{5}(-0.46) = 0.09
\end{aligned}
$$

Based on the above calculations, the rules are ordered by their gain. Finally, if the applied rule fails (i.e., when zero or more than one elements are identified while a single value property is expected, e.g. for the case of a title), the next rule is selected. For the example in Table 1, the following rules are produced in the following order:[8]

1. CSS Class for value $cssValue1$
2. CSS Class for value $cssValue2$
3. HTML Identifier for value $id1$
4. Absolute path for $path1$
5. Absolute path for $path2$
6. Absolute path for $path3$

[7]Since null contains no information, we consider instances of null values as a unique case which does not add up to a more general case of "null" values.

[8]Observations from the evaluation presented in Section 6 show that the use of CSS Classes or HTML Identifiers is prioritised over the Absolute Path attribute, when the information gain values are equal.

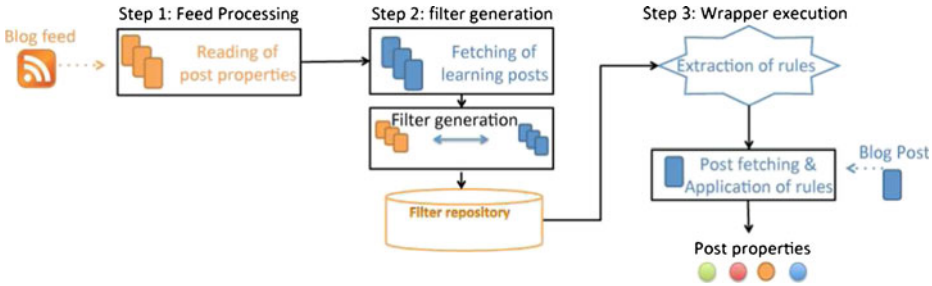


Figure 2 Overview of the weblog data extraction methodology

Figure 2 presents an overview of the overall approach described. As already discussed in detail, the proposed solution involves the execution of three steps. The first step includes the task of reading and storing the weblog data properties found in the web feed. The second step includes training the wrapper through the cross matching of information found in the web feed and the corresponding HTML documents. This step leads to the generation of information, captured through the filters, which describes where the weblog data properties reside. The final step deals with the processing of the filters, in order to generate the rules that are used for weblog data extraction.

4 Text matching

As already discussed in Section 3.2, the proposed method relies on the identification of an HTML element against a specific value. Text matching can be used for achieving the above identification. Generally, text matching is a research field studied extensively and may be classified into the following string matching cases.

- *Complete* matching: Every character of a string has to match every character of another string and vice versa.
- *Partial* matching: Every character of one string matches a substring of another string. Insertion is needed to have a complete matching.
- *Absolute* matching: A sequence of characters matches completely another one.
- *Approximate* matching: A sequence of characters is aligned to another sequence optimally; i.e., the least possible changes take place in order to have an absolute matching. Substitution is needed to achieve absolute matching.

Whilst absolute matching is a fast and trivial task, this type of matching is of limited interest and applicability. On the contrary, approximate matching has been a field of study in several problem domains. The main reason for the popularity of approximate string matching is that inherent data distortion appears in many problems.[9] For the case under examination (identifying specific HTML elements) approximate, partial text matching[10] is needed, since two different, semi-structured data sources (the HTML document and the web feed) are cross-matched (see Table 2 for an example).

[9] A detailed survey on various string matching techniques can be found at [8].

[10] also used as "optimal string alignment".

Table 2 An example of different cases of text matching (taken and extended from http://en.wikipedia.org/wiki/Levenshtein_distance)

	Absolute	*Approximate*
Complete	kitten	kitten
	kitten	sitten
Partial	kitten	kitten
	kitt	sitting

The Levenshtein (or edit) distance is quite popular because it presents several advantages (see Table 3). First of all, it is a simple concept and its return value can be effortlessly assessed as a measure of the conducted matching/comparison. Secondly, the algorithm that calculates this distance is very efficient computationally (time and space complexity are low) and finally, this distance establishes a *metric space*.[11] However, the Levenshtein distance value expresses the number of character edits and therefore is not unitless. Thus, having an absolute value as a result of the matching between two strings does not give much information about the "degree of matching" between them. For example, comparing one string of 200 characters against another of 1000 characters would lead in the best case scenario (complete, partial matching) to a Levenshtein distance value of 800. At the same time, comparing two strings of 200 bytes would yield a result of no more than 200 (worst case scenario, no overlapping characters between the two strings). The above remarks have led to the further development of text distance measurements. Typically, these metrics adopt and extend the Levenshtein distance while at the same time attempt to address the issue of strings of different length. The most common variations are the ones presented in Table 4.

It is worth noting that each variation presents some advantages over the original Levenshtein distance. For example, both d_{sum} and $d_{\max}$ have normalised values in the scale of 0 to 1 and are unitless (i.e. do not express number of characters but percentage). The same applies to the distance d_{YB}, which has the extra feature of establishing a metric space. On the other hand, $d_{\min}$ is not normalised and therefore maintains the basic limitation of the original Levenshtein distance. However, experiments conducted during the evaluation.[12] have shown that all four distance metrics fail to quantify partial matching of weblog properties adequately for the cases of strings of notably different length (see discussion in previous paragraph).

Taking into account the above, another distance metric that is not based on the Levenshtein metric was considered; the Jaro-Winkler metric, which is introduced by Winkler [26] and extends the metric by Jaro [14]. Jaro metric is computed as [8]:

$$Jaro(\sigma_1, \sigma_2) = \frac{1}{3}\left(\frac{c}{|\sigma_1|} + \frac{c}{|\sigma_2|} + \frac{c - t/2}{c}\right)$$

where:

– $|\sigma_1|$ and $|\sigma_2|$ are the lengths of strings σ_1 and σ_2;

[11] Establishing a metric space based on a distance function leads to a very powerful mathematical model. This is because a metric space presents important properties (e.g. the triangular inequality, identity of indiscernibles, symmetry) that can be used to extract meaningful knowledge between objects belonging to a large collection, without the need to calculate the distance for every pair of objects.

[12] Evaluation conducted for text matching is omitted, since it exceeds the scope of this paper and would increase the size to a great extent.

Table 3 Levenshtein distance values for the example of Table 2. Distance is measured by the minimum number of character edits between two strings

Pairs of strings	Levenshtein distance
(kitten, kitten)	0
(kitten, sitten)	1
(kitten, kitt)	2
(kitten, sitting)	3

– c are the "common characters" in the two strings: common are all the characters $\sigma_1[i]$ and $\sigma_2[j]$ for which $\sigma_1[i] = \sigma_2[j]$ and $|i - j| \leq \frac{1}{2}min\{|\sigma_1|, |\sigma_2|\}$;
– find the number of transpositions t; the number of transpositions is computed as follows: compare the ith common character in σ_1 with the ith common character in σ_2, and each nonmatching character is a transposition.

Winkler and Thibaudeau [26] modified the Jaro metric to give higher weight to prefix matches, which for our case favours the matching of clipped post content found in web feeds. Our evaluation of the above metrics has shown that the Jaro-Winkler distance is more appropriate, since it returns low values for cases of partial matching between strings of considerably different lengths.

5 Example

In order to demonstrate how the proposed methodology works, we consider a specific example of a weblog. The weblog shown here is hosted on http://blogs.warwick.ac.uk, which is powered by the blogging platform BlogBuilder, developed by Warwick IT services.[13] The weblog of the example provided is owned by a user by the name of "John Smith".[14] For this example, the property to be extracted will be the author of the weblog. The steps are as follows:

Step 1: Feed processing and capturing of post properties

In this step, the web feed was found and processed. More specifically, the feed is an Atom feed[15] containing entries for the latest 20 posts published by the user. However, in order to keep the example as general possible, the latest 10 posts were considered as being part of the web feed and the other 10 were discarded (10 is the most typical number of entries found in web feeds [21]). For this feed, the value of the author is always the same (John Smith); however, even if more than one author was blogging, steps would proceed in the same manner, since the value to be matched is extracted on a per-post, case-by-case basis.

Step 2: Generation of filters

In this step, the value of the author found in the feed is matched to the HTML document of the weblog post. For each match, a filter is generated, described by the attributes HTML

[13] Our evaluation presented in Section 6 uses a great variety of weblogs. The selection of one Warwick blog is simply for illustrative purposes.

[14] The name of the user has been anonymised for privacy reasons.

[15] http://tools.ietf.org/html/rfc4287

Table 4 Variations of the Levenshtein distance (taken by Yujian and Bo [29]). $|x|$, $|y|$ are the lengths of the strings x,y, respectively. d_E is the Levenshtein (or edit) distance

Method Name	*Expression*
Normalised by the sum	$d_{sum}(x, y) = \frac{d_E(x,y)}{\|x\|+\|y\|}$
Normalised by the maximum of the length of the strings	$d_{\max}(x, y) = \frac{d_E(x,y)}{\max\{\|x\|,\|y\|\}}$
Normalised by the minimum of the length of the strings	$d_{\min}(x, y) = \frac{d_E(x,y)}{\min\{\|x\|,\|y\|\}}$
Normalisation by Yujian and Bo [29]	$d_{YB}(x, y) = \frac{2d_E(x,y)}{\|x\|+\|y\|+d_E(x,y)}$

Identifiers, CSS Classes and Absolute Path accordingly. The filters extracted are shown in Table 5.

The first column of Table 5 refers to a unique identifier of each post. The table shows that, for each of the 10 posts, at least one filter was generated, which means that the value of the author found in the web feed was successfully matched for every case. Moreover, we notice that there are cases (i.e. post ids 18113 and 18115) where the author appears more than once (2 and 3 times, respectively). This is happening because the text "John Smith" either appears in a separate paragraph (as a "signature" to the content of the post 18113) or as a comment author (post 18115). However, for the majority of the cases the author name is embedded into the HTML as follows:

```
<span class="author">
<img src="/blogbuilder/media/images/tiny_author.gif"
title= "2nd Year Undergraduate Student:
Statistics" alt="">
John Smith </span>
```

The filters presented in the Table 5 indicate that the HTML Identifiers are not used to identify the HTML element (all values are null), while the CSS Class value *author* is used with high frequency. Finally, for the Absolute Path, while it follows a common path to a certain extent, its value typically varies.

Step 3: Induction of rules and weblog data extraction

In this step, the gain for each value of each attribute is calculated, as described in Section 3. The result of the calculation shows that the first rule is the one where value *author* is selected for the case of CSS Classes (highest gain). Moreover, if we attempt to evaluate this rule, the result will show that the author is extracted successfully for all posts of the weblog (a more thorough discussion on evaluation issues follows in Section 6). The specific values for each rule are shown in Table 6.

6 Evaluation

In order to assess and evaluate the described methodology, we have developed a software prototype that implements it. The evaluation was based on a use case that aimed to assess the accuracy of the weblog data extraction. Details about this use case are the following.

– A collection of 2547 posts (originating from 325 weblogs) was created. These weblogs are powered by different blogging engines, such as Blogger or WordPress and use

different themes. The selected blogs are the most popular (in terms of incoming links) as indicated by the Spinn3r dataset [5]. For each weblog, the feed containing a list of posts was acquired.
– Each weblog feed contained at least 10 entries. If more entries were included, the additional entries were discarded and were not used for training or evaluating the wrapper (see explanation in the previous section).
– The weblog post properties *title*, *author*, *publication date* and *post content* were evaluated.

Evaluation was conducted following the *10-fold validation* method [28]. This method is most appropriate when evaluating the quality of a classification approach. The aim of this method is to assess the expected accuracy of future examples, a measurement known as *prediction error*. In order to achieve the above, training data are partitioned into 10 disjoint folds (blocks). Training takes place for 9 out of the 10 blocks and testing of the model occurs for the remaining block. This task is repeated 10 times, in order to test all blocks

Table 5 Filters generated from and for a single weblog, for the case of a specific author, John Smith. Attribute HTML Identifiers is null for all cases and is omitted

Post Id	CSS Classes	Absolute Path
18110	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/ div[0]/div[1]/div[1]/div[0]/p[8]/span[0]
18111	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/ div[0]/div[1]/div[1]/div[0]/p[13]/span[0]
18112	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/ div[0]/div[1]/div[1]/div[0]/p[7]/span[0]
18113	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/ div[0]/div[0]/div[1]/div[1]/div[0]/p[9]/span[0]
18113	Ø	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/ div[0]/div[0]/div[1]/div[1]/div[0]/p[7]
18114	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/ div[0]/div[1]/div[1]/div[0]/p[2]/span[0]
18115	commentauthor	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/ div[0]/div[1]/div[1]/div[1]/div[0]/ol[2]/li[0]/h4[5]
18115	commentauthor	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/ div[0]/div[0]/div[1]/div[1]/div[1]/div[0]/ol[2]/li[0]/h4[5]/a[0]
18115	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/ div[0]/div[1]/div[1]/div[0]/p[2]/span[0]
18116	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/ div[0]/div[0]/div[1]/div[1]/div[0]/p[7]/span[0]
18117	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/ div[0]/div[0]/div[1]/div[1]/div[0]/p[7]/span[0]
18118	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/ div[0]/div[0]/div[1]/div[1]/div[0]/p[6]/span[0]
18119	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/ div[0]/div[0]/div[1]/div[1]/div[0]/p[7]/span[0]

Table 6 Gain for every attribute value of the filters of Table 5. Selecting the rule described by the highest gain (CSS Class with value *author*) will result in the successful extraction of the desired property for all weblog posts. Attribute values appearing once have been omitted for brevity

Attribute name	Attribute value	Occurrences	Gain
CSS Classes	author	10	0.22
Absolute Path	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[7]/span[0]	4	0.16
CSS Classes	commentauthor	2	0.06
Absolute Path	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[2]/span[0]	2	0.06

(see Figure 3). In our case, each block corresponds to an entry found in a web feed. Finally, the prediction error is extracted by accounting the errors for all 10 cases.

The evaluation was conducted on the author and the title of a weblog post. While the web feed of the weblog contains more information, like the content and the publication date of the post, validation of the correct extraction of these properties was omitted. For the case of the content, the feed contains less information, namely the summary. For the case of the date, the styling of the date during the training was not captured and it was therefore not easy to cross-validate the extraction. An updated version of the prototype includes more capabilities concerning the date matching in order to be included in future validations. Given the above, a manual validation on a small sample of the use case has shown that the proposed approach captures the correct data in most of the cases and is equally effective for the properties which have been 10-fold validated.

The results overall show that accuracy for the 2547 weblog posts is high (89 %). For the case of the title, the accuracy is as high as 98.3 % (43 misses). For the case of the author, the accuracy drops to 80.6 % (493 misses). A survey on the errors of the author attribute shows that these errors appear mostly due to the following reasons.

- There are cases where a weblog does not show the name of the author at all. Instead, the author name might appear occasionally as an author of a comment, leading to the false production of rules.
- Similarly, there are cases where the author name appears as a comment in addition to the post author. This leads to the generation of several filters that point to comment blocks, leading to an increased information gain of elements found in comments.

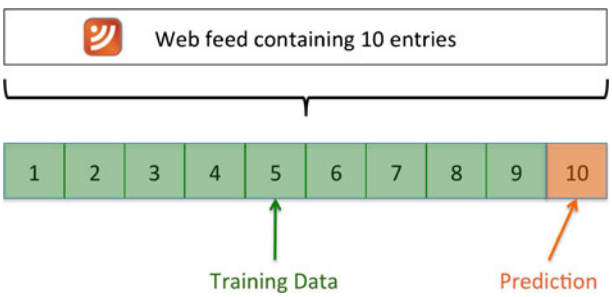


Figure 3 A visualisation of the 10-fold validation approach applied

– Because of the time difference between the fetching of the web feed and the weblog post, there are cases where weblog posts have changed (false negative).

During the development, some issues surfaced that shaped the final proposed extraction methodology, which have not been discussed yet. These issues, as well a suggestion for further improvement, are as follows.

– Matching a date (e.g. publication date) is a complex task, compared to the simpler text matching (e.g. the weblog post title). The main reason for this is the fact that different weblogs may style the date in different ways. For instance, a date might appear as "Friday, 11 May 12", "Friday, 11 May 2012" or even 11/05/2012. In order to address the above issue, the prototype currently casts the date value to the most popular styles (as well as some variations of them) available, such as:

    – SHORT (e.g. 05.11.12)
    – MEDIUM (e.g. May 11, 2012)
    – LONG (e.g. May 12, 2012)
    – FULL (e.g. Friday, May 11, 2012 AD)

  Moreover, the date is currently rendered in the English language only, which is usually not sufficient for matching the date in weblogs written in different languages. An improvement would be to identify the language of the document (e.g. with Tika[16]) and style the date following the locale of the language identified.
– Initially, the rules are ordered by their information gain. If a rule fails to return a valid value (e.g., a rule for the title should return exactly one HTML element, but the rule might return 0 or more than one elements), the next rule is applied until the model runs out of rules. This is not always the best practice, since there are cases where searching for a property should actually return no value (i.e. a title being blank). In this particular case, the current prototype exhausts all possible rules, leading to false values. An improvement would be to select rules that present information gain higher than a certain threshold. We are currently experimenting with various configurations and data in order to further study the above settings.
– For the case of the content of the weblog post, a web feed often contains a small snippet of the actual post. This is the reason why the Jaro-Winkler distance measurement was selected over the variations of the Levenshtein distance. Therefore, after the summary has been identified in the HTML document, the summary is extended in order to generate the filter for the full content. The software prototype addresses this issue as follows:

    – while the parent element does not contain special tag names (e.g. div), the algorithm considers the parent element as the placeholder for the content recursively;
    – when a special tag name is found (e.g. div), the recursion stops and the element is used to generate the filter.

Based on lessons learnt from the above, more recent work has been carried out and an updated technique which follows and extends the same methodology has been developed [12]. The new technique leverages the advantages of its predecessor (presented in this paper) and overcomes some of the limitations mentioned above. Future work includes the

[16] http://tika.apache.org/

experimentation with more rich XPath expressions as extraction rules in order to increase the performance and allow applying the proposed technique beyond weblogs.

7 Discussion and related work

This section discusses how our approach relates to similar ones. To make the section more readable, we are splitting the discussion into three parts. The first part presents techniques that are making use of web feeds for accommodating data extraction (*Web Feed Based Extraction*). The second part places our approach in the general research framework of data extraction (*Data Extraction Techniques*). The last part discusses how our approach can be integrated into spiders and enhance their performance (*Crawling and Accessing Weblog Data*).

7.1 Web feed based extraction

The concept of using web feeds for capturing data is not new. ArchivePress[17] is one of the archiving projects that has developed solutions for harvesting the content of weblog feeds. Their solution focuses solely on collecting the structured content of weblog feeds that contain posts, comments and embedded media. ArchivePress offered a robust mechanism for extracting weblog content and rich metadata. Making use of the standard XML-based data format has enabled much more accurate and easier retrieval of data compared to using a traditional crawler such as Heritrix.[18] Despite these benefits, the solution provided by ArchivePress remains highly limited. The first limitation is related to the low number of entries (i.e., ten entries on average) distributed via a single feed. Relying solely on web feeds prevents archiving data that are no longer available in feeds. The second limitation is bound to encountering feeds that distribute only partial content (i.e., post summary) as opposed to full content. Therefore, retrieval when using ArchivePress is accurate but incomplete. Finally, the offered solution, as well as other related plugins, can only be deployed on a single weblog at a time (powered by the WordPress blogging platforms only and customised with the provided plugin)[19] and does not provide the necessary coverage for extracting information from a large number of weblogs.

Despite the fact that web feeds constitute a useful resource for extracting information from weblogs, they remain insufficient for ensuring robust information extraction. Hence, the consideration of HTML documents with weblog data remains paramount. One of the approaches that attempts to combine the two was developed by Oita and Sellenart [21]. The authors describe an approach for identifying the main article by using their corresponding web feeds. This approach is based on evaluating a web page of interest and matching it to the semantic information found in the corresponding web feed. The general principle of cross-matching web feeds and pages constitutes the foundation of the approach that we proposed in this paper. However, because the approach by Oita and Sellenart does not devise general extraction rules, the solution proposed is not suitable for capturing the data that are no longer available in the corresponding web feed. This is a considerable limitation that prevents the adoption of the proposed approach for extracting all the weblog entries, including those that

[17] http://archivepress.ulcc.ac.uk/

[18] http://crawler.archive.org/

[19] http://code.google.com/p/archivepress

are no longer distributed via web feeds. The approach introduced in our paper enables to overcome this limitation by describing a mechanism for devising a set of general extraction rules that can be used across weblog entries and regardless of their presence in the web feed. Oita and Sellenart [21] report the results of an evaluation of a small corpus of weblogs as comparable to other approaches; however, the performance of their approach for extracting distinct properties, such as title, was reported as poor and was not discussed in detail.

7.2 Data extraction techniques

The review of related work demonstrates the gap in the area of web information extraction and highlights the need for developing robust solutions that are applicable for weblogs. This paper attempts to narrow this gap by introducing a novel approach that enables capturing the content along with the semantic structure of weblogs. To position our approach within the domain of earlier conducted work on web information extraction, we classify it according to the taxonomy of data extraction tools by Laender [17]. The rationale for conducting this classification is to support potential users in assessing the suitability of adopting the approach within a more general context of information extraction. The taxonomy consists of six groups of tools. The grouping of the tools is performed by taking into account their affordances: degree of automation, support for complex objects, ease of use, XML output, support for non-HTML sources and type of page contents. The approach proposed in this section can be associated with the Wrapper Induction and Modelling-Based groups. Similarly to the Wrapper Induction tools, our approach generates extraction rules as discussed in Section 3.3 of this paper. However, unlike many wrapper induction tools, our approach is fully automated and does not rely on human-annotated examples. Instead, it uses web feeds as a model that informs the process of generating extraction rules and also resembles the Modelling-Based group. Hence, the approach presented in this paper can be positioned in relation to tools such as WIEN [16], Stalker [20], RoadRunner [6] or NoDoSE [1].

WIEN is among the first tools aimed at automating the process of information extraction from web resources. The term *wrapper induction* is, in fact, coined by the authors [16] of the tool. However, as one of the earlier attempts, the use of the tool is restricted to a specific page structure and the heuristics of the presented data. Furthermore, it is not designed to work with nested structures of web data. The limitation of working with hierarchical data has been addressed by the Stalker tool [20]. However, the use of Stalker requires a training data set that limits the degree of automation offered by the system. An attempt to automate the process of wrapper induction was made by Crescenzi et al. [6] and published along with the RoadRunner tool, which analyses structurally similar resources and generates a common schema for extracting the data. NoDoSE [1] represents a different, modelling-based category of tools that requires an existing model that defines the process of extraction. This is a semi-automatic approach due to the necessary human input for developing models. However, additional tools, such as a graphical user interface for marking resources, can be used for reducing the human workload of developing the models. Hence, the review of the earlier work suggests that the approach proposed in this paper addresses a niche not served by the existing tools.

Among the generic solutions there are other technologies that aim at identifying the main section (e.g., article) of a web page. The open source Boilerpipe[20] system is state-of-the-art and one of the most prominent tools for analysing the content of a web page [15]. Boilerpipe

[20] http://code.google.com/p/boilerpipe/

makes use of the structural features such as HTML tags or sequences of tags forming subtrees and employs methods that stem from quantitative linguistics. Using measures, such as average word length and average sentence length, Boilerpipe analyses the content of each web page segment and identifies the main section by selecting the candidate with the highest score. The use of Boilerpipe delivers relatively good results. For instance, the evaluation by Oita and Sellenart [21] reports the precision of Boilerpipe to be 62.5 %, even though our experiments with weblogs suggest a higher performance. However, similarly to Heritrix, the main limitation of adopting Boilerpipe is bound to the constraints in identifying the semantic structure of a weblog and in ensuring higher granularity of the identified content that distinguishes properties such as *title*, *date* and *author* from the *content* of the post.

7.3 Crawling and accessing weblog data

The approach proposed in this paper can be adopted for designing and developing specialised spiders for the Blogosphere. Berger et al. [4] highlight the gap for developing solutions for information extraction from weblogs and discuss the potential of employing web feeds for capturing weblog data. However, the authors do not elaborate on their approach and focus on the process of designing a weblog crawler and discussing its architecture at a higher level. Hence, the existing gap for introducing effective weblog information extraction remains to be bridged. The approach introduced in our paper narrows this gap. In other work, Heritrix, the open source crawler used by the Internet Archive,[21] allows for revisiting webpages and identifying important changes using certain heuristics [24]. The above solutions share a common strategy, where the attempt is to pairwise compare webpages in order to automatically identify the template that renders the webpage. This approach resembles the generic process of a wrapper induction. However, while the use of Heritrix may identify the template of the weblog, this approach does not capture the semantic structure of the weblog as represented through the template. Finally, Faheem and Senellart propose an Application-Aware Helper (AAH) that can be used during the crawling process [9]. AAH is using a knowledge base that searches for HTML patterns and manages to detect the underlying blogging engine. AAH and similar website crawling strategies are extremely useful before the execution of our model in order to provide the target post-pages.

In agreement with AAH, we performed an analysis on the total collection of the attributes of the filters (HTML Identifiers and CSS Classes) generated during the evaluation of Section 6. This analysis has revealed some patterns that may be used in order to train the model faster and increase its accuracy. The patterns show that the most common values for the HTML Identifiers of the author are “header” and “site-title,branding,header”. For the CSS Classes and the case of author or title, the values appearing most frequently are shown in Table 7. As part of a future work, the values in Table 7 may be used in our model in order to promote several well-known values and discard others (e.g., the value *comment-author* is used to describe a comment and not the author of a post, even if there are cases where the values are identical) or even better inform a crawling component, such as AAH’s knowledge base.

[21] http://www.archive.org/

Table 7 Some values of CSS Classes ordered by the number of occurrences. The first column corresponds to the case of the author and the second to the title of a weblog post

Author	*Title*
fn	entry-title
author,vcard	post-title
url,fn,n,author,vcard	title
url,fn	storytitle
url	posttitle
comment-author	post-format-icon
url,comment-author	pagetitle
author	entrytitle
url,fn,n	single-title
comment-author,vcard	rsswidget
fn,comment-author,vcard	single
url,fn,comment-author,vcard	photo-title

8 Conclusions

In this paper, we have presented a method for fully automated weblog wrapper generation. Compared to state-of-the-art tools, the generated wrapper exhibits increased granularity, since it manages to identify and extract several weblog properties, such as the *title*, *author*, *publication date* and *main content* of the post with a prediction accuracy of 89 %. This is accomplished through the generation of rules, which are selected through the adoption of a probabilistic approach based on information entropy and gain. The devising of these rules is based on the generation of filters. The filters constitute a structure that, when applied to a web document, singles out an HTML element. They are described in triples, where each of its element-attributes describes the HTML element in different forms (Absolute Path, CSS Classes and HTML identifiers). The overall approach is evaluated against a real-world collection of weblogs and the results show that the wrappers generated are robust and efficient in handling different types of weblogs.

Acknowledgments This work was conducted as part of the BlogForever project funded by the European Commission Framework Programme 7 (FP7), grant agreement No.269963.

References

1. Adelberg, B.: NoDoSE-a tool for semi-automatically extracting structured and semistructured data from text documents. SIGMOD Rec. **27**(2), 283–294 (1998)
2. Baumgartner, R., Flesca, S., Gottlob, G.: Visual web information extraction with lixto. In: Proceedings of the 27th International Conference on Very Large Data Bases, VLDB '01, pp. 119–128. Morgan Kaufmann Publishers Inc., San Francisco (2001)
3. Baumgartner, R., Gatterbauer, W., Gottlob, G.: Web data extraction system. In: Encyclopedia of Database Systems, pp. 3465–3471. Springer (2009)
4. Berger, P., Hennig, P., Bross, J., Meinel, C.: Mapping the blogosphere–towards a universal and scalable blog-crawler. In: Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE Third Int Confernece Soc Comput (SocialCom), pp. 672–677. IEEE (2011)
5. Burton, K., Kasch, N., Soboroff, I.: The ICWSM 2011 Spinn3r dataset. In: Proceedings of the Fifth Annual Conference on Weblogs and Social Media (ICWSM 2011). Barcelona, Spain (2011)

6. Crescenzi, V., Mecca, G., Merialdo, P.: Roadrunner: towards automatic data extraction from large web sites. In: Proceedings of the International Conference on Very Large Data Bases, pp. 109–118 (2001)
7. Dutton, W., Blank, G.: Next generation users: The Internet in Britain. Oxford Internet Survey. http://www.oii.ox.ac.uk/publications/oxis2011_report.pdf (2011)
8. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007)
9. Faheem, M., Senellart, P.: Intelligent and adaptive crawling of web applications for web archiving. In: Proceeding ICWE. Aalborg (2013)
10. Geibel, P., Pustylnikov, O., Mehler, A., Gust, H., Kühnberger, K.: Classification of documents based on the structure of their DOM trees. In: Neural Information Processing, pp. 779–788. Springer (2008)
11. Giles, K., Bryson, K., Weng, Q.: Comparison of two families of entropy-based classification measures with and without feature selection. In: Proceedings of the 34th Annual Hawaii International Conference on System Sciences, HICSS '01, p. 3014. IEEE Computer Society, Washington (2001)
12. Gkotsis, G., Stepanyan, K., Cristea, A., Joy, M.: Self-supervised automated wrapper generation for weblog data extraction. In: G. Gottlob, G. Grasso, D. Olteanu, C. Schallhart (eds.) Big Data, Lecture Notes in Computer Science, vol. 7968, pp. 292–302. Springer, Berlin (2013)
13. Ihara, S.: Information theory for continuous systems. World Scientific Publishing Company, Singapore (1993)
14. Jaro, M.A.: Unimatch: A Record Linkage System: User's Manual. Tech. rep., U.S. Bureau of the Census, Washington DC (1976)
15. Kohlschütter, C., Fankhauser, P., Nejdl, W.: Boilerplate detection using shallow text features. In: Proceedings of the 3rd ACM International Conference on Web Search and Data Mining, WSDM '10, pp. 441–450. ACM, New York (2010)
16. Kushmerick, N.: Wrapper induction: efficiency and expressiveness. Artif. Intell. **118**(1), 15–68 (2000)
17. Laender, A., Ribeiro-Neto, B., Da Silva, A., Teixeira, J.: A brief survey of web data extraction tools. ACM Sigmod Rec. **31**(2), 84–93 (2002)
18. Liu, B.: Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data. Springer-Verlag, Berlin (2009)
19. Liu, L., Pu, C., Han, W.: XWrap: an extensible wrapper construction system for internet information. In: Proceedings of the 16th International Conference on Data Engineering (ICDE 2000), pp. 611–621. IEEE CS Press, San Diego (2000)
20. Muslea, I., Minton, S., Knoblock, C.: Hierarchical wrapper induction for semistructured information sources. Auton. Agent. Multi-Agent Syst. **4**(1), 93–114 (2001)
21. Oita, M., Senellart, P.: Archiving data objects using web feeds. In: Proceedings of International Web Archiving Workshop, pp. 31–41. Vienna, Austria (2010)
22. Pennock, M., Davis, R.: ArchivePress: a really simple solution to archiving blog content. In: Sixth International Conference on Preservation of Digital Objects (iPRES 2009). California Digital Library, San Francisco (2009)
23. Quinlan, J.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
24. Sigurthsson, K.: Incremental crawling with Heritrix. In: Proceedings of International Web Archiving Workshop, pp. 1–12 (2005)
25. Web Technology Survey: Usage of content management systems for websites. [Online]. Available: http://web.archive.org/web/20131015180119, http://w3techs.com/technologies/overview/content_management/all. Accessed Oct 2013, Tech. rep., W3Techs (2013)
26. Winkler, W.E., Thibaudeau, Y.: An application of the fellegi-sunter model of record linkage to the 1990 us decennial census. Methods **9** (1990)
27. Winn, P.: State of the Blogosphere 2008: Introduction. http://technorati.com/blogging/article/state-of-the-blogosphere-introduction/ (2009). Accessed 21 Aug 2009
28. Witten, I.H., Frank, E.: Data Mining. Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann (2005)
29. Yujian, L., Bo, L.: A normalized levenshtein distance metric. IEEE Trans. Pattern Anal. Mach. Intell. **29**(6), 1091–1095 (2007)
30. Zhai, Y., Liu, B.: Web data extraction based on partial tree alignment. In: Proceedings of the 14th International Conference on World Wide Web, pp. 76–85. ACM (2005)