CALA: An unsupervised URL-based web page classification system

Inma Hernández [a,*], Carlos R. Rivero [b], David Ruiz [c], Rafael Corchuelo [c]

[a] Universidad Autonoma de Chile, Carlos Antunez 1920, Santiago, Chile
[b] Department of Computer Science, University of Idaho, 875 Perimeter Drive, MS 1010, Moscow, ID 83844-1010, United States
[c] Universidad de Sevilla, ETSI Informática, Avda. de la Reina Mercedes, s/n, Sevilla E-41012, Spain

Keywords:

Web page classification
URL classification
URL patterns
Enterprise web information integration
Web page clustering

A B S T R A C T

Unsupervised web page classification refers to the problem of clustering the pages in a web site so that each cluster includes a set of web pages that can be classified using a unique class. The existing proposals to perform web page classification do not fulfill a number of requirements that would make them suitable for enterprise web information integration, namely: to be based on a lightweight crawling, so as to avoid interfering with the normal operation of the web site, to be unsupervised, which avoids the need for a training set of pre-classified pages, or to use features from outside the page to be classified, which avoids having to download it. In this article, we propose CALA, a new automated proposal to generate URL-based web page classifiers. Our proposal builds a number of URL patterns that represent the different classes of pages in a web site, so further pages can be classified by matching their URLs to the patterns. Its salient features are that it fulfills all of the previous requirements, and it has been validated by a number of experiments using real-world, top-visited web sites. Our validation proves that CALA is very effective and efficient in practice.

1. Introduction

Web page classification refers to the problem of assigning a web page a class that describes its contents, which has many interesting applications: enterprise web information integration [25], helping crawlers find pages about a given topic [13,52], choosing the most appropriate model to extract information from a web page [16,28,50], parental control systems [55], removing advertisements [48], or detecting and canonising duplicated URLs [3,34], to mention a few.

Our focus is on enterprise web information integration. This is a general term that refers to a variety of techniques whose goal is to provide a unified view over the data provided by a number of web applications; the ultimate goal is to allow retrieving information from these applications as if they were a database. The majority of such applications provide a keyword-based search form [26,37]. To integrate the information they deliver, it is necessary to use wrappers [42]. A wrapper is a piece of software that endows a web site with a search form with a programmatic interface that allows executing queries on it. The execution of a query involves filling and submitting the form, which returns a response page. If the response page contains the information of interest to answer the query (that is, the query was precise enough, e.g., "ISBN 1590338022", or "Samsung Galaxy Note 10.1 WiFi N8010"), then an information extractor can be immediately applied to extract that information [15,51,46,49]; otherwise (that is, the query was not precise enough, e.g., "Java", or "Tablet"), it is common that web sites return a hub page, which provides listings of links to different pages that may have information of interest. In the latter case, it is necessary to use a navigator [1,41] to analyse the links in the hub and determine which ones need to be followed. Web page classifiers [44,45] play a fundamental role to implement navigators, since they allow discerning programmatically between both types of pages.

There are many techniques to learn a web page classifier. They all require to download a subset of web pages, which is commonly referred to as training set, and analyse some of their features. These features can be external to the pages (e.g., features in their URLs or the context where they are linked), or internal (e.g., words or HTML structure). Some techniques are supervised, i.e., they require a person to pre-classify the pages in the training set, but others are unsupervised, i.e., their results must be interpreted by a person.

For a learning or classification technique to be useful in the context of enterprise web information integration, it must fulfill three requirements:

(R1) Lightweight crawling: To learn a classifier it is necessary to provide a training dataset. Such a dataset is gathered by performing a crawl of the site being analysed. Such a crawl should be as lightweight as possible since it should not interfere with

* Corresponding author. Tel.: +34 95 455 27 70; fax: +34 95 455 71 39.
E-mail addresses: ichernandez@uautonoma.cl (I. Hernández), crivero@uidaho.edu (C.R. Rivero), druiz@us.es (D. Ruiz), corchu@us.es (R. Corchuelo).

the normal operation of the site. Furthermore, web sites change frequently [20,33] and it is not uncommon that these changes render the classifier obsolete. Therefore, the classifier must be learnt not once but several times and performing an extensive crawling that covers a large portion of a web site becomes unfeasible.

(R2) Unsupervision: It is important that a person does not need to pre-classify each page in the training set individually, since this is an effort-consuming and error-prone task; neither should he or she provide any additional training information. In other words, it is important that the techniques used to learn a classifier be unsupervised or, otherwise, they shall not scale well to the Web [38].

(R3) Classify without downloading: Downloading a web page puts a load on the server, takes time, and consumes bandwidth. That is, it is important that a classifier relies exclusively on external features of a page in order to classify it, since it would be inefficient for practical purposes otherwise [31].

Requirements R1 and R2 are related to the process of learning a classifier, whereas R3 is related to the classification of web pages using that classifier.

The literature provides a variety of techniques to learn web page classifiers; they can be broadly classified into the following categories: Contents-based proposals [7,29,36,47], structure-based proposals [16,4,2,12,18,53], visual-based proposals [56,19,35], link-based proposals [11,17,22,57,54], and URL-based proposals [14,31,6,34,52,3,13]. The proposals in the first three categories rely on internal features, i.e., they require to download a web page prior to analysing its contents, structure, or visual features; this makes them of little interest regarding enterprise web information integration since they do not fulfill our third requirement. The proposals in the fourth category build on analysing the graph of links amongst the pages of a web site; thus, they require to perform an extensive crawling in order to learn a classifier, which does not fulfill our first requirement. The proposals in the fifth category rely on features of the URLs, which, in principle, makes them more appropriate for enterprise web information integration; unfortunately, we review them all in Section 2 and our conclusion is that none of them satisfies the three previous requirements at a time, which motivated us to work on a new technique called CALA (ClAssifying Links Automatically).

CALA is a proposal to learn a web page classifier that does not require a previous extensive crawling, it is unsupervised, and it does not require a page to be downloaded so that it can be classified. The system relies on two modules, namely: a crawler, which gathers a small set of hubs from a web site in order to assemble a training set; and a pattern builder, which uses the previous training set to build a set of patterns, each of which represents a collection of URLs that are expected to reference web pages of the same class. We have performed an experimental analysis and we have compared the results to two well-known techniques in the literature: Template Page Model (TPM) by Blanco et al. [12] and Support Vector Clustering (SVC) by Ben-Hur et al. [8]. Regarding effectiveness, we achieved precision and recall values that are statistically similar to the results of TPM and SVC, whereas the $F1$ values are better than the results of both techniques. Regarding efficiency, our timings are similar to those of TPM and considerably faster than those of SVC.

We presented a two-page abstract of our proposal elsewhere [27]. In this article, we extend this abstract with an in-depth description of the algorithms and an experimental analysis that proves that they are both efficient and effective. The rest of this article is organised as follows: Section 2 presents the related work regarding URL-based web page classification; Section 3 defines our proposal to build URL patterns for web page classification; Section 4 shows the evaluation of our technique; finally, Section 5 concludes the article.

2. Related work

In this section, we analyse the proposals in the literature to learn URL-based web page classifiers. A nanve approach is to use clustering techniques; they rely on a distance function and return a number of clusters that verify that the inter-distance is maximum, whereas the intra-distance is minimum. Since URLs can be naturally represented as strings, the idea would be to use a string distance. Unfortunately, it has been noticed that using classic string distances does not work well to classify URLs [13] since two close URLs may provide information about two different classes, whereas distant URLs may be related to web pages of the same class. For example, assume that <MSAS> is an abbreviation for the domain name academic.research.microsoft.com. There is a minimum distance between URLs http://<MSAS>/Detail?entitytype=2&searchtype=2&id=35096884 and http://<MSAS>/Detail?entitytype=1&search-type=5&id=35096884, but they reference web pages that are likely to be classified in different classes (publications of an author and citations made to that author's papers). Contrarily, URLs like http://<MSAS>/Author/2542366/charles-antony-richard-hoare and http://<MSAS>/Author/10540585/yu-li are far more distant but belong to the same class (authors). It remains unexplored whether using non-classic distances might improve the results.

Beyond the previous nanve approach, other authors have proposed a number of ad hoc techniques that are summarised in Table 1. In the following paragraphs, we provide additional details on each proposal.

Brin [14] presented DIPRE, a supervised proposal to extract structured information from web pages. It considers the Web as a database of unstructured information, and it aims at gathering tuples from it (e.g., books). This proposal takes a set of sample tuples as input, and it performs an incremental process that consists of the following steps: first, it looks for occurrences of the sample tuples in the Web, i.e., it looks for web pages where the attributes of one of the tuples occur near to each other. For each occurrence, the URL of the web page on which it appears and the text that surrounds it are considered the context of the occurrence. Afterwards, DIPRE uses these contexts to generate patterns that match occurrences with a similar context. These patterns include a URL prefix, which is the longest common prefix to the URLs of the occurrences, and a text pattern, which is a regular expression that matches the text surrounding the occurrences. Finally, it looks for tuples in the Web matching the new patterns. The process iterates until enough patterns have been generated. Note that DIPRE requires performing an extensive crawling of the Web to gather as many tuples of the target relation as possible.

Table 1
Comparison of current URL-based proposals. (R1 = Lightweight crawling; R2 = Unsupervision; R3 = Classify without downloading).

Proposals	R1	R2	R3
Brin [14]	No	No	Yes
Kan and Thi [31]	No	No	Yes
Vidal et al. [52]	No	No	Yes
Bar-Yossef et al. [3]	No	Yes	Yes
Baykan et al. [6]	No	No	Yes
Koppula et al. [34]	No	Yes	Yes
Baykan et al. [5]	No	No	Yes
Blanco et al. [13]	No	Yes	No
CALA	Yes	Yes	Yes

Kan and Thi [31] proposed a supervised web page classifier for pages in different web sites that is based exclusively on features computed from their URLs. The URLs are tokenised using the standard RFC 3986 format for URIs, and their tokens are used as features; then, more features are computed, such as the position of each token in the URL, the length of the URL, or the lexical kind of token (e.g., if it represents a number, a word, or a non-alphabetical symbol). These features are used as input to an entropy maximisation algorithm, a well-known machine learning approach that is usually applied to text classification [9,43]. To build the classifier, they use large training sets of URLs to achieve good precision and recall, which requires a previous extensive crawling of the sites that are being analysed.

Baykan et al. [6,5] presented a supervised web page classification proposal that builds exclusively on URLs. They create feature vectors by tokenising URLs and then use those features to build a support vector machine and a nanve-bayes classifier. In their experiments, they use large training sets of URLs, and they require the user to provide a list of words and URLs that are representative of every class; furthermore, they also require a sample set of URLs that are not representative of each class.

Vidal et al. [52] proposed a supervised technique to classify web pages building on their URL. Their proposal takes a sample page as input, and returns a set of URL patterns that match the URLs of pages that are structurally similar to the sample page. It is based on two steps: site mapping and pattern generation. Site mapping consists in building a map of the web site, which requires to crawl the entire site starting from its home page and following every possible path. They keep a record of the paths in the map that lead (directly or indirectly) to pages that are similar to the sample page. The similarity is measured using a tree-edit distance between the DOM trees underlying the pages. Then, pattern generation consists of generalising the URLs of the pages in the former paths using regular expressions, and then selecting the path that leads to the largest number of target pages.

Bar-Yossef et al. [3] proposed a supervised technique to detect web pages with different URLs that have the same contents, which has a negative impact on crawling efficiency. To solve this problem, they classify URLs according to the contents of their target, and they build regular expressions to define each class of URLs. Then, those URLs are normalised using a rule mining algorithm. They need to have a large collection of URLs to achieve good results, which means that a previous extensive crawling of the web site must be performed to gather them. A similar proposal was presented by Koppula et al. [34].

Blanco et al. [13] proposed an unsupervised algorithm to classify web pages that combines external features and optional internal features. Their proposal is based on the idea that every web site is created by populating a number of HTML templates with data from a database, and that the URLs of those pages are created by populating a URL template with data from the same database. Therefore, pages created from the same HTML template have similar contents and URLs generated from the same URL template link to pages with similar contents. They proposed an algorithm that combines web page contents and its URL as features to cluster web pages so that each cluster contains pages that were created using a certain template. Their algorithm is based on the well-known minimum description length method [23]. They require a large training set, so they crawl the entire site in their experiments. Note that to improve the classification efficiency, internal features can be used, which means that in some cases the page must be downloaded previously.

From the previous paragraphs, we conclude that none of the techniques in the literature fulfills the three requirements we have identified at a time. Our proposal does, since it does not require the web site to be crawled extensively, but only a small set of hubs, it is totally unsupervised, and it does not require to download a page to classify it.

3. Our proposal

In this section, we present our proposal: first, we introduce some concepts that we use throughout the article; then, we present our crawler and our pattern builder. Throughout this article, we use a mathematical notation that is based on the Z specification [30]. Furthermore, we use Microsoft Academic Search as a running example; it is a scholarly web site that offers information about items that include papers, authors, citations, and publishing hosts, such as journals or conferences. This web site offers a keyword-based search form in which a person can write the keywords that describe the items in which he or she is interested, as depicted in Fig. 1.

3.1. Preliminaries

Our proposal relies on the analysis of the URLs of the web pages provided by a number of hubs. Generally speaking, a URL describes the access protocol and the location of a resource. According to Berners-Lee et al. [10], a URL like http://<MSAS>/Detail?entitytype=1&searchtype=5&id=48814179#stats is composed of the following segments: first, a protocol (http); then, an authority or domain name (<MSAS>); afterwards a sequence of path segments separated by slash characters (Detail); and two optional sections: a question mark symbol followed by a query string, and/or a sharp symbol followed by a fragment. A query string provides information about the names and the values of a number of parameters sent to the web server (entitytype=1&searchtype=5&id=48814179 includes parameters entityType, searchType, and id with values 1, 5, and 48814179, respectively). Finally, the fragment is a sequence of characters that indicates a specific section inside a page (stats).

We define a token as a subsequence of characters that is delimited by separators ://, /, ?, &, =, and #. In other words, a token is a string of characters that denotes a protocol, a path segment, a parameter, a value of a parameter, or a fragment. We introduce URLs and tokens as three given sets in our framework since we do not need to delve into their structure:

$$[URL,\ Token]$$
$$Separator == \{: //, /, ?, \&, =, \#\}$$

We define a pattern as a sequence of tokens, separators, and wildcards that starts with a $\hat{}$ symbol and ends with a $ symbol. A wildcard, which we denote as ★, is a placeholder that accounts for any token. Note that given a URL, it is straightforward to tokenise it into a pattern; thus, we do not provide any additional details on this procedure.

A prefix refers to a subsequence of a pattern that starts at the first token and extends up to another token in the pattern. Note that a pattern is similar to a prefix, since both of them are sequences of tokens, separators, and wildcards; the only difference between them is that a pattern ends with a $. We formally define the previous concepts as follows:

$$Marker == \{\bigstar, \hat{}, \$\}$$
$$Prefix == \{p : \mathrm{seq}(Token \cup Separator \cup Marker) \mid \#p > 1 \land p(1) = \hat{} \land (\forall i : \mathbb{N} \mid i > 1 \land i \leqslant \#p - 1 \cdot p(i) \notin \{\hat{}, \$\}) \land \mathrm{last}\ p \neq \hat{}\}$$
$$Pattern == \{p : Prefix \mid \mathrm{last}\ p = \$\}$$

A hub page is a web page that results from submitting a keyword-based search form using some keywords as query, and provides links to other web pages [32]. Note that hub pages usually contain a larger number of URLs than other pages since their goal

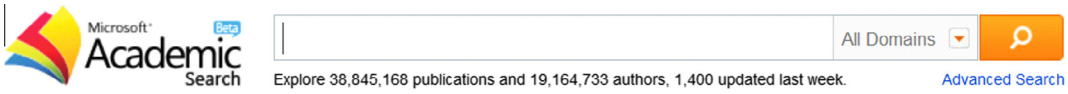

Fig. 1. Search form page in the running example.

is to offer as many results related to the queries as possible. Regarding our proposal, a hub can be abstracted as a set of patterns that result from the URLs in the links it provides. A hubset is a collection of hubs that result from submitting a search form using different keywords. We formally define the previous concepts as follows:

$$[Word,\ WebPage]$$
$$Hub == \text{seq}\ Pattern$$
$$Hubset == \text{seq}\ Hub$$

The last preliminary concept comes from statistics. Our crawler and pattern builder require to discern hubs and tokens whose size or frequency, respectively, deviate largely from the mean values. These values are commonly referred to as outliers and they can be identified very easily using Cantelli's inequality. Based on this inequality, we define a lower threshold and an upper threshold below or above which the values of a set can be considered lower or upper outliers, respectively. The thresholds rely on a given confidence level, which is usually referred to as α. We formally define these thresholds as follows:

$$lowerThreshold : \text{seq}\,\mathbb{R} \rightarrow \mathbb{R}$$
$$upperThreshold : \text{seq}\,\mathbb{R} \rightarrow \mathbb{R}$$

$$\forall R : \text{seq}\,\mathbb{R} \mid R \neq \emptyset \bullet$$
$$\text{let}\ \mu == \text{mean}\ R; \sigma == \text{stdev}\ R; k == \sqrt{(1-\alpha)/\alpha} \bullet$$
$$lowerThreshold(R) = \mu - k\,\sigma$$
$$upperThreshold(R) = \mu + k\,\sigma$$

3.2. Our crawler

Our crawler is responsible for retrieving a set of hubs from a web site, using the URL of a page with a keyword-based search form as input. The crawler is based on the algorithm in Fig. 2. It takes the URL of a page with a keyword-based search form as input and outputs a hubset. We assume that the following constants have been set before executing this algorithm: *M*, which refers to the number of hubs the algorithm is expected to return; *T*, which refers to the maximum number of attempts that the algorithm is allowed to make in order to gather *M* hubs; and *N*, which refers to the number of keywords that we select from each page.

We make the following conjectures regarding these constants (these conjectures and others that we establish throughout this article are corroborated by our experimental results in Section 4.4):

Conjecture 1 (*Value of M*). *A relatively small number of hubs suffices to achieve a high precision and recall in our proposal.*

Conjecture 2 (*Value of T*). *A relatively small number of attempts suffices to gather M hubs.*

Conjecture 3 (*Value of N*). *A relatively small number of keywords from each page suffices to gather M hubs.*

In the following subsections, we describe the ancillary functions on which the algorithm relies.

```
1: algorithm  gatherHubset
2: input      u : URL
3: output     Result : Hubset
4: variables  wp, hp : WebPage; P, Q : seq Word; R : seq WebPage, n : ℕ
5: constants  M, T : ℕ
6:
7: n := 0
8: P := ∅
9: R := ∅
10: Result := ∅
11: – Step 1: Download initial page
12: wp := download(u)
13: – Step 2: Compute keywords from the page contents
14: Q := computeKeywords(wp, P)
15: while n < T ∧ Q ≠ ∅ ∧ #R < M do
16:     while Q ≠ ∅ ∧ #R < M do
17:         – Step 3: Submit the form in wp using a keyword from Q
18:         kw := get one keyword from Q
19:         P := P ∪ {kw}
20:         Q := Q \ {kw}
21:         hp := submit(wp, kw)
22:         – Step 4: Add new page to set
23:         R := R ∪ {hp}
24:         – Step 5: Update the set of keywords
25:         Q := Q ∪ computeKeywords(hp, P)
26:     end while
27:     – Step 6: Keep only non-empty hubs
28:     R := getNonEmptyHubs(R)
29:     n := n + 1
30: end while
31: for each r ∈ R do
32:     – Step 7: Create a new hub using the patterns in r
33:     Result := Result ∪ {computePatterns(r)}
34: end for
```

Fig. 2. Algorithm to gather hubs starting from a initial page with a keyword-based search form.

3.2.1. Computing keywords

Function *computeKeywords* takes a web page and a set of words as input, and returns a set with the *N* least frequent words in the page that are not included in the set. We formally define this function as follows:

$$computeKeywords : WebPage \times \text{seq}\ Word \rightarrow \text{seq}\ Word$$

$$\forall wp : WebPage, P : setWord \bullet$$
$$\text{let}\ K == \text{subseq}((\text{sortBag}\ computeWords(wp) \setminus P), 1, N) \bullet$$
$$computeKeywords(wp, P) = \{k : Word \mid k \in K\}$$

To compute the frequencies of each word in *wp*, we compute the bag of words using function *computeWords*, and then we sort them according to their frequency using function sortBag. Note that *P* denotes the set of keywords so far processed in Algorithm *gatherHubset* and that it is passed on to this function as a parameter to check if any of the *N* least frequent keywords has already been used. This way, only the keywords that have not been considered before and have a low frequency are considered.

In our running example, we gathered the following keywords from the page in Fig. 1: authors, Advanced, Search, publications, last, updated, and, week, and Explore, all of which appear once in the page.

3.2.2. Discarding empty hubs

Empty hubs are usually very similar: they display an image and/or a message to inform the person that his or her query did not

produce any results, that it was incorrect, or that an unrecoverable error happened, and, optionally, a few links to recommended items. This implies that their size, in terms of number of patterns, tends to be smaller than usual and that they can be discarded by looking for lower outliers. We formally define this function as follows:

$$getNonEmptyHubs : Hubset \pfun Hubset$$

$$\forall hs : Hubset \mid hs \neq \emptyset \bullet \\ \quad \text{let } t == lowerThreshold(\{h : Hub \mid h \in hs \bullet \#h\}) \bullet \\ \quad\quad getNonEmptyHubs(hs) = \{h : Hub \mid h \in hs \land \#h \geq t\}$$

In our running example, non-empty hubs have sizes of around 130.43 patterns with a standard deviation of 25.43 patterns; using the standard $\alpha = 0.05$, we get a lower threshold of 19.58, i.e., every hub with less than 20 patterns is considered to be empty.

3.2.3. Other ancillary functions

Our crawler relies on a few more functions that are straightforward, but difficult to present within a formal framework. We describe them below:

download : *URL* → *WebPage*. This function takes a URL as input and returns a copy of the document therein located. A typical web page usually requires a number of resources so that it can be rendered, e.g., images, style sheets, or scripts. These resources are not necessary at all in our proposal, so this function can discard them safely and return the HTML text only.

computeWords : *WebPage* → bag *Word*. This function takes a web page as input and returns a bag with the words in that page, excluding tags, scripts, and in-line style definitions. In order to keep our proposal language-independent, we used the following unicode regular expression to implement this function: ((\p{L}\p{N}+)|\p{N}+\p{L})|\p{L})+, where \p{L} is a regular expression that represents unicode characters that are letters and \p{N} represents unicode characters that are numbers.

submit : *WebPage* × *Word* → *WebPage*. This function takes a web page with a keyword-based search form, and a word as input. It is responsible for finding the keyword-based search form, filling its unique text field using the word, submitting it, and returning the response hub page.

computePatterns : *WebPage* → *Hub*. This function takes a web page as input and returns a hub with the patterns in that page, which are extracted from its links.

```
1: algorithm  buildPatterns
2: input      hs : Hubset
3: output     Result : seq Pattern
4: variables  P, S, S', W, W' : seq Prefix; p, q : Prefix; pe, t : ℝ
5:
6: Result := ∅
7: – Step 1: Initialise prefix set
8: P := initialisePrefixSet(hs)
9: while P ≠ ∅ do
10:    p := shortest prefix in P
11:    if last p = $ then
12:       P := P \ {p}
13:       Result := Result ∪ {p}
14:    else
15:       – Step 2: Compute siblings
16:       S := computeSiblings(hs, p)
17:       – Step 3: Compute upper-outlying siblings
18:       t := upperThreshold({q : Prefix | q ∈ S • p-estimator(hs, q)})
19:       S' := ∅
20:       for each q ∈ S do
21:          pe := p-estimator(hs, q)
22:          if pe ≤ t ∧ pe < 1.00 − β then
23:             S' := S' ∪ {q}
24:          end if
25:       end for
26:       – Step 4: Wildcard prefixes
27:       W := {v, w : Prefix | v ∈ S' ∧ w ∈ P ∧ v prefix w • w}
28:       W' := wildcardPrefixes(W, #p)
29:       – Step 5: Update prefix set
30:       P := updatePrefixSet(P, S, W, W')
31:    end if
32: end while
```

Fig. 3. Algorithm to mine URL patterns.

3.3. Our pattern builder

The pattern builder relies on the algorithm in Fig. 3. It takes a hubset *hs* as input and outputs a set of patterns that represent the different classes of pages in the input site.

In our running example, algorithm *buildPatterns* outputs four patterns, cf. Table 2. In the following subsections, we describe the ancillary functions on which the algorithm relies.

3.3.1. Initialising prefix set

Function *initialisePrefixSet* takes a hubset as input and returns a sequence with the prefixes of the patterns in flat *hs*, ordered in increasing order of size. We formally define this function as follows:

$$initialisePrefixSet : Hubset \to \text{seq } Prefix$$

$$\forall hs : Hubset \bullet \\ \quad initialisePrefixSet(hs) == \\ \quad\quad \text{sortSet}\{p : Prefix \mid \#p \geq 2 \land (\exists q : Pattern \mid q \in \text{flat } hs \bullet p \text{ prefix } q)\}$$

We use a tree notation that is based on PATRICIA trees to represent sets of prefixes; it allows representing a large collection of patterns and prefixes compactly. Every node in the tree has a label, which we denote as n_i or $s_i, i \in \mathbb{N}$, and a token. Each node, actually, represents a prefix, which is the sequence of tokens and separators in the path from the tree root n_0 to the node itself. Note that each path from the tree root to a leaf represents a pattern. From now on, in the examples, we refer to prefixes in the tree by means of the corresponding node: n_i for tokens or s_i for separators.

As an example, Fig. 4 represents a subset of the prefixes in our running example. Note that it is not possible to show the complete set since it has thousands of prefixes.

3.3.2. Computing siblings

Function *computeSiblings* takes a hubset *hs* and a prefix *p* as input and returns the set of prefixes in flat *hs* that share a common prefix with *p* up to its penultimate element, including *p* itself. We formally define this function as follows:

$$computeSiblings : Hubset \times Prefix \pfun \text{seq } Prefix$$

$$\forall hs : Hubset, p : Prefix \mid hs \neq \emptyset \land \#p \geq 2 \bullet \\ \quad computeSiblings(hs, p) = \{q : Prefix; r : Pattern \mid \\ \quad\quad r \in \text{flat } hs \land q \text{ prefix } r \land \text{last } q \neq \star \land (\text{front } p) \text{ prefix } q \bullet \text{subseq}(q, 1, \#p)\}$$

In our running example, prefix $n_4 = \langle$, http, ://, <MSAS>, /, Publication, /, 4117664$\rangle$ has siblings $n_6 = \langle$, http, ://, <MSAS>, /, Publication, /, 5638047$\rangle$ and $n_8 = \langle$, http, ://, <MSAS>, /, Publication, /, 1242380$\rangle$. In Fig. 4, it is easy to find the siblings of a prefix represented by a node n_i by looking for nodes that share an ancestor with n_i.

3.3.3. Computing p-estimators

Function *p*-estimator takes a hubset *hs* and a prefix *p* as input and returns an estimator of the probability of finding at least one pattern prefixed by *p* in a hubset. Since the underlying distribution of prefixes is unknown and heavily dependent on the web site

Table 2
Patterns built for the running example.

Pattern	Class
⟨∧, http, ://, ⟨MSAS⟩, /, Publication, /, ★, /, ★, $⟩	Paper
⟨∧, http, ://, ⟨MSAS⟩, /, Author, /, ★, /, ★, $⟩	Author
⟨∧, http, ://, ⟨MSAS⟩, /, Journal, /, ★, /, ★, $⟩	Journal
⟨∧, http, ://, ⟨MSAS⟩, /, Detail, ?, entityType, =, 1, &, searchType, =, 5, &, id, =, ★, $⟩	Citation

being analysed, we can compute a *p*-estimator as the relative frequency of every prefix in *hs* [24]. We formally define the calculation of a *p*-estimator as follows:

$$p\text{-}estimator : Hubset \times Prefix \rightarrow \mathbb{R}$$

$$\forall hs : Hubset, p : Prefix \mid hs \neq \emptyset \bullet \\ \text{let } H == \{h : Hub \mid h \in hs \wedge (\exists q : Pattern \bullet q \in h \wedge p \text{ prefix } q)\} \bullet \\ p\text{-}estimator(hs, p) = \#H / \#hs$$

p-Estimators range from 0.00 to 1.00. The more frequent a prefix, the higher its corresponding *p*-estimator. We state the following conjecture regarding *p*-estimators:

Conjecture 4 (*Distribution of p-estimators*). *p-Estimators that are not near 1.00 are most probably near* $1/\#hs$*, whose limit is 0.00 as the number of hubs increases. In other words, the distribution of p-estimators has two peaks at 0.00 and 1.00.*

In our running example, the *p*-estimators computed for the prefixes in Fig. 4 are shown in Table 3. As an example, note that prefix $n_3 = \langle$, http, ://, < MSAS >, /, Publication$\rangle$ appears in every hub in *hs* since this prefix refers to publications and every hub in our running example includes links to publications. Therefore, its *p*-estimator is 1.00. Every hub includes a list of publications, and each publication has at least one author, which means that we can find at least one pattern prefixed by n_3 in every hub. Contrarily, some publications are published in journals and some others are published in conferences, which means that only some of the hubs contain patterns prefixed by n_{19}. Therefore, the *p*-estimator of prefix $n_{19} = \langle$, http, ://, < MSAS >, /, Journal$\rangle$ is not 1.00, but 0.95 in this case; this estimator is significantly high, but not as high as the *p*-estimator of prefix n_3.

3.3.4. Wildcarding prefixes

Function *wildcardPrefixes* takes a set of prefixes and a natural number *i* as input, and it returns a set in which the input prefixes have been wildcarded at the *i*th position, i.e., the token at this position has been changed into a wildcard of the form ★. We formally define this function as follows:

$$wildcardPrefixes : \text{seq } Prefix \times \mathbb{N} \rightarrow \text{seq } Prefix$$

$$\forall C : \text{seq } Prefix; i : \mathbb{N} \mid i \geq 1 \bullet \\ wildcardPrefixes(C, i) = \{p : Prefix \mid p \in C \wedge \#p \geq 2 \wedge i \leq \#p \bullet \\ \text{subseq}(p, 1, i-1) + \langle \star \rangle + \text{subseq}(p, i+1, \#p)\}$$

In our running example, the *p*-estimator of prefix $n_4 = \langle$, http, ://, <MSAS>, /, Publication, /, 4117664$\rangle$ and its siblings $n_6 = \langle$, http, ://, <MSAS>, /, Publication, /, 5638047$\rangle$ and $n_8 = \langle$, http, ://, <MSAS>, /, Publication, /, 1242380$\rangle$ is 0.01 according to Table 3. Since they all have the same value, their mean value is 0.01, and their standard deviation is 0.00. Therefore, the upper threshold is 0.01 in this case. Since none of the siblings has a *p*-estimator higher than the threshold, they are added to the set of prefixes to be wildcarded, including the prefixes that have them as a prefix, namely $s_4, s_5, s_6, n_5, n_7, n_9, n_{34}, n_{35}$, and n_{36}. Fig. 5(b) shows the siblings of n_4 and their descendants after wildcarding them. Note that we change the name of node n_4 to w_2 to emphasise the fact that the final tokens in prefixes n_4, n_6, and n_8 have been replaced with a wildcard, i.e., they three have become a single new prefix that is represented by node w_2. The descendants of prefixes n_4, n_6, and n_8 now share the same prefix w_2.

Fig. 5(a) presents the tree with the resulting prefix set after executing algorithm *buildPatterns* on the running example. Some of the prefixes in the tree are the result of wildcarding one or more of the original prefixes using function *wildcardPrefixes*. Note that every path from the tree root to a leaf represents a different pattern.

3.3.5. Updating the prefix set

Function *updatePrefixSet* takes four sets of prefixes as input: P, S, W, and W'. In each iteration of algorithm *buildPatterns*, these sets represent, respectively, the ordered prefix set in algorithm *buildPatterns*, the set of siblings that have been processed, the set of prefixes to be wildcarded, and another set in which the prefixes

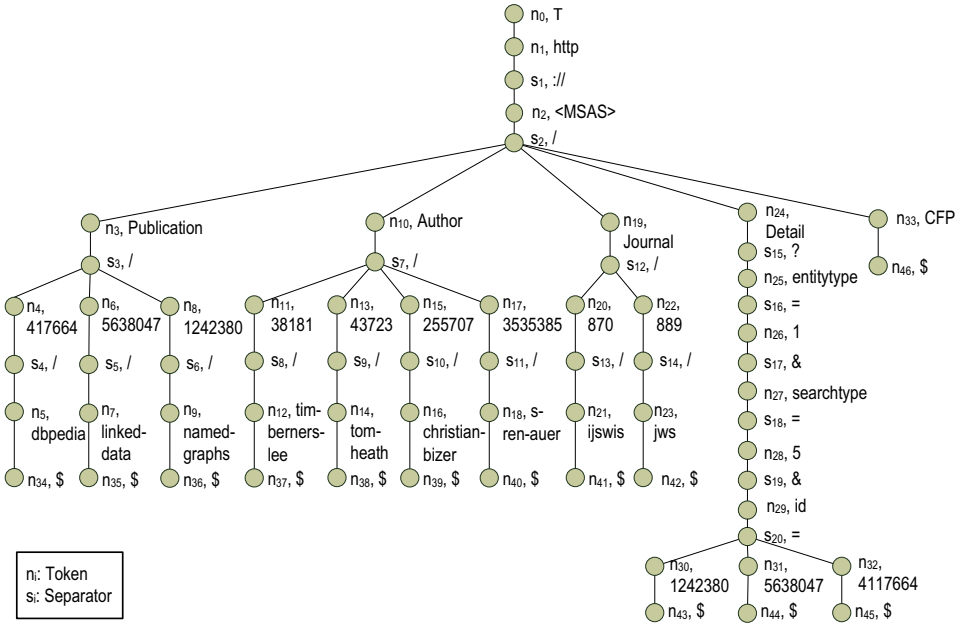

Fig. 4. Tree with a subset of the initial prefix set in the running example.

Table 3
p-Estimators in the running example.

Node	Last token	*p*-Estimator
n_0	^	1.00
s_1	://	1.00
n_4	417664	0.01
s_4	/	0.01
n_8	1242380	0.01
s_6	/	1.00
s_8	/	1.00
n_{13}	43723	0.02
s_{10}	/	1.00
n_{17}	3535385	0.01
s_{12}	/	1.00
n_{20}	870	0.01
n_{23}	jws	0.01
n_{24}	Detail	1.00
s_{17}	&	1.00
n_{27}	searchType	1.00
s_{20}	=	1.00
n_{31}	5638047	0.01
n_1	http	1.00
n_3	Publication	0.99
s_3	/	0.99
s_5	/	1.00
n_7	linked-data	0.01
s_7	/	1.00
n_{11}	38181	0.01
n_{14}	tom-heath	0.02
n_{15}	255707.00	0.01
n_{18}	s-ren-auer	0.01
n_{19}	Journal	0.95
n_{21}	ijswis	0.01
s_{14}	/	1.00
s_{16}	=	1.00
n_{26}	1.00	1.00
s_{19}	&	1.00
n_{29}	id	1.00
n_{32}	4117664	0.01
n_2	⟨MSAS⟩	1.00
s_2	/	1.00
n_5	dbpedia	0.01
n_6	5638047	0.01
n_9	named-graphs	0.01
n_{10}	Author	0.99
n_{12}	tim-berners-lee	0.01
s_9	/	1.00
n_{16}	christian-bizer	0.01
s_{11}	/	1.00
s_{13}	/	1.00
n_{22}	889	0.01
s_{15}	?	1.00
n_{25}	entityType	1.00
s_{18}	=	1.00
n_{28}	5	1.00
n_{30}	1242380.00	0.01
n_{33}	CFP	0.17

have already been wildcarded. The function returns a prefix set that is the result of updating P by replacing the prefixes to be wildcarded with their wildcarded version, and subtracting the sibling prefixes that have been processed in that iteration. We formally define this function as follows:

$$updatePrefixSet : \text{seq } Prefix \times \text{seq } Prefix \times \text{seq } Prefix \rightarrow \text{seq } Prefix$$

$$\forall P, S, W, W' : \text{seq } Prefix \bullet updatePrefixSet(P, S, W, W') = \text{sortSet}((P \setminus W \setminus S) \cup \{q : Prefix \mid q \in W' \setminus S\})$$

4. Experimental evaluation

In this section, we report on our experimental evaluation. Our goal is threefold: first, we aim to prove that our technique is able to classify web pages with good precision and recall with regard to other baseline classification techniques; second, we aim to prove that our technique is very efficient with regard to other techniques; finally, we aim to corroborate the conjectures on which our proposal relies.

4.1. Dataset definition

We have carried out our experimentation with a collection of datasets that we chose from the Alexa top 40 web sites that were written in English and provided a keyword-based search form. Since the Alexa ranking[1] changes daily, we fixed a reference date to select the sites (February 14, 2011). The dataset included four additional academic sites, namely: TDG Scholar, Google Scholar, Microsoft Academic Search, and Arxiv. From each site, we downloaded 100 hubs to perform the experiments, i.e., we had 4400 pages available for experimentation. These datasets are available at the authors' web site.[2]

For each web site, we identified the classes that best described their pages. For instance, in Amazon, we identified the following classes: Product, Author, and Review. We did not include every possible class of pages from each site in the analysis, since some of the classes were of little interest for the user (e.g., error pages, disambiguation pages, advertisement pages, external pages, and the like); instead, we included just those classes that more accurately classified the different types of contents of interest that were offered by each site.

Since we had 4400 pages, we decided to use a technique to automate the labelling of the pages: for each hub, we used a Firefox plug-in[3] to identify XPaths of the links that led to each of the classes that we selected for every site. Note that it's relatively simple for a person to classify a page building on the information that a hub shows. The labelling took roughly 100 work hours, whereas we estimated that labelling each page individually would have taken more than 1000 work hours, not to mention that this would have led to many classification errors and would have been difficult to scale. The XPath expressions that we identified are available at the author's web site.[4]

4.2. Evaluation design

For the configuration of our crawler, we set the parameters to the following values: $M = 100, T = 5$, and $N = 10$, i.e., we gathered 100 hubs from each site, made a maximum of 5 attempts to download them, and selected 10 keywords from each hub. Other things equal, increasing M, T, or N did not have an impact on the effectiveness of our proposal; however, decreasing M, T, or N had a negative impact when analysing a few sites.

For each dataset, we used our pattern builder and evaluated its effectiveness. We used 50% of the hubs in each dataset to infer a set of patterns (training set) and the remaining 50% to validate our proposal (test set). We set both the confidence level for calculating outliers α and the similarity threshold β to 0.05. Regarding the confidence level, recall that it defines the fraction of data in a distribution that are considered outliers; we selected the standard value in the literature. Regarding the similarity threshold, recall that it is a small value that allows considering two close numbers equal; there is not a standard in the literature, but we found out through repeated experimentation that setting it to other values had a slight negative impact on the effectiveness of our proposal. The same experiments were repeated using our baselines using

[1] http://www.alexa.com/topsites.
[2] http://www.tdg-seville.info/inmahernandez/Experiment.
[3] https://addons.mozilla.org/en-US/firefox/addon/xpath-checker/.
[4] http://tdg-seville.info//Download.ashx?id=398.

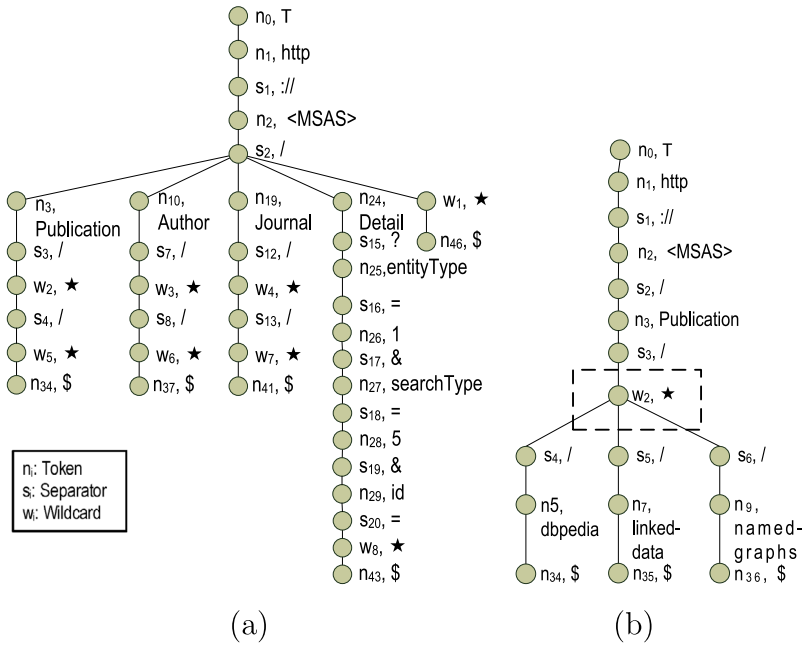

Fig. 5. Prefixes in our running example after pattern mining.

Table 4
Results of the evaluation. *P* = Precision; *R* = Recall; *F*1 = F1-measure; *T* = CPU learning time (s).

Summary		CALA				TPM				SVC			
		P	R	F1	T	P	R	F1	T	P	R	F1	T
Mean		0.98	0.90	0.92	9.02	1.00	0.65	0.70	4.17	0.66	0.94	0.73	3483.99
Std. deviation		0.09	0.19	0.18	10.66	0.00	0.39	0.38	5.81	0.32	0.18	0.27	8230.65
Site	Class	P	R	F1	T	P	R	F1	T	P	R	F1	T
Amazon	Products	1.00	0.94	0.97	34.06	1.00	0.46	0.63	16.78				
	Reviews	1.00	0.99	1.00		1.00	0.22	0.36					
	Authors	1.00	1.00	1.00		1.00	1.00	1.00					
DailyMotion	Videos	1.00	1.00	1.00	2.78	1.00	0.50	0.67	1.53	0.74	1.00	0.85	613.94
	User Profiles	1.00	1.00	1.00		1.00	0.91	0.95		0.23	1.00	0.37	
Ehow	Articles	1.00	0.99	0.99	2.06	1.00	0.00	0.00	0.86	1.00	1.00	1.00	269.28
Answers	Topics	0.92	0.99	0.95	5.06	1.00	0.50	0.67	6.03	0.23	1.00	0.38	12922.00
	Questions	1.00	1.00	1.00		1.00	0.98	0.99		0.76	1.00	0.86	
Digg	Authors	0.99	1.00	1.00	3.80	1.00	0.49	0.66	6.59	0.24	1.00	0.39	392.89
	Articles	1.00	1.00	1.00		1.00	0.00	0.00		0.36	1.00	0.53	
	Comments	1.00	1.00	1.00		1.00	0.33	0.49		0.37	1.00	0.54	
Indiatimes	Articles	1.00	0.45	0.62	5.42	1.00	0.00	0.00	0.53	0.99	0.70	0.82	1240.00
DailyMail	Authors	1.00	1.00	1.00	9.48	1.00	0.50	0.67	7.19	0.11	0.98	0.19	49386.00
	Articles	1.00	0.45	0.62		1.00	1.00	1.00		0.43	1.00	0.60	
Deviantart	Photos	1.00	1.00	1.00	8.86	1.00	1.00	1.00	4.44	0.86	1.00	0.92	7133.13
	Tags	0.94	0.31	0.47		1.00	0.50	0.67		0.12	0.96	0.21	
Filestube	Files	1.00	0.94	0.97	7.97	1.00	1.00	1.00	0.34	0.92	1.00	0.96	69.33
Huffingtonpost	Articles	1.00	0.18	0.31	3.73	1.00	0.83	0.90	5.30	1.00	0.99	1.00	145.42
Sourceforge	Projects	1.00	1.00	1.00	8.02	1.00	0.96	0.98	0.84	0.62	1.00	0.76	438.84
	Reviews	1.00	1.00	1.00		1.00	0.49	0.66		0.36	1.00	0.53	
Squidoo	Articles	1.00	0.99	1.00	2.08	1.00	0.00	0.00	2.27	0.60	1.00	0.75	96.52
	User Profiles	1.00	1.00	1.00		1.00	0.49	0.66		0.35	1.00	0.52	
Torrentz	Files	1.00	1.00	1.00	3.33	1.00	1.00	1.00	0.17	0.98	1.00	0.99	1172.50
Guardian	Authors	0.65	1.00	0.99	11.17	1.00	1.00	1.00	7.55	0.58	1.00	0.73	1206.13
	Articles	0.35	1.00	0.02		1.00	0.87	0.93		1.00	0.64	0.78	
Archive	Articles	1.00	0.98	0.99	13.28	1.00	0.98	0.99	0.23	0.98	1.00	0.99	728.22
Isohunt	Files	1.00	0.97	0.98	4.94	1.00	0.98	0.99	2.69	0.48	0.96	0.64	5400.06
	Comments	1.00	1.00	1.00		1.00	1.00	1.00		0.49	1.00	0.66	
Yelp	Businesses	1.00	0.79	0.88	7.59	1.00	1.00	1.00	24.61	1.00	1.00	1.00	263.58
Metacafe	Videos	1.00	0.76	0.86	4.20	1.00	0.25	0.40	4.41	0.50	1.00	0.67	1316.25
	Topics	1.00	0.97	0.99		1.00	1.00	1.00		0.27	0.99	0.42	
	User Profiles	1.00	0.91	0.95		1.00	0.48	0.65		0.20	1.00	0.33	
Etsy	Products	1.00	0.95	0.97	21.55	1.00	0.52	0.69	2.78				
	Stores	1.00	1.00	1.00		1.00	0.42	0.60					

also a half of the hubs as the training set and the other half as the test set.

In Section 2, we compared our system with other proposals in the related work from a theoretical point of view. We did not find any available implementation of the techniques described in Section 2, so we had to resort to the following state-of-the-art classification techniques:

Template Page Model Classification Scheme, or TPM for short [12]: It is a supervised structure-based web page classifier, that detects common templates in a set of training pages. We implemented this proposal in the lab using Java. The proposal relies on a similarity threshold that was set to 0.75 following the authors' recommendation. We chose TPM to prove that our unsupervised technique is as effective as a state-of-the-art technique without requiring the effort to provide a pre-classified training set, and without having to download a page to classify it.

Support Vector Clustering, or SVC for short [8]: It is an unsupervised URLbased clustering technique that is based on Support Vector Machines. We chose this clustering technique because it does not require setting a number of clusters beforehand and it is particularly suitable for clustering web pages since it can deal with large sets of classification features; in this context, each feature corresponds to a token in a pattern and counts its frequency in the datasets. We used a modified version of RapidMiner [40] in which we implemented a validation module; we used the suggested parameter values: radial kernel with $\gamma = 1.00$, a kernel cache of size 200, a minimum of 2 pages per cluster, a convergence of 0.001, and a maximum of 100,000 iterations. We chose SVC to prove that our proposal performs better than a state-of-the-art unsupervised technique building solely on URL-based features.

The experiments were run on a cloud computer that was equipped with a four-threaded 64-bit 2.93 GHz Intel i7 processor, 16 GiB of RAM, Oracle Java Development Kit 1.6.0_25, and Windows 7 Pro 64-bit.

4.3. Evaluation results

Tables 4 and 5 report on the results of our experiments. The columns report on the precision (P), recall (R), the $F1$ measure $\left(F1 = 2\frac{P\ R}{P+R}\right)$, and the learning time in CPU seconds (T). The first two rows provide a summary of these measures in terms of mean values and standard deviations. A dash in a cell means that the corresponding technique was not able to learn a classifier in one week's time.

Regarding P and R, we used a technique by [39] to calculate them in a non-supervised fashion; that is, we did not assume an a priori correspondence between each URL pattern and one of the classes. Instead, intermediate precision and recall were calculated for every possible combination of pattern/class, and the final precision and recall were the weighted means of the intermediate values, where the weight was the number of URLs that the pattern and class of each combination had in common. By doing so, we were taking into account the possibility that our patterns included URLs from more than one class of pages, and vice versa.

We were able to draw the following conclusions: regarding effectiveness, CALA outperforms the other techniques regarding

Table 5
Results of the evaluation (Cont.) P = Precision; R = Recall; $F1$ = F1-measure; T = CPU learning time (s).

Summary		CALA				TPM				SVC			
		P	R	F1	T	P	R	F1	T	P	R	F1	T
BBC	News	1.00	0.57	0.73	3.66	1.00	0.00	0.00	2.31	0.37	0.98	0.54	12.33
	Videos	1.00	1.00	1.00		1.00	0.00	0.00		0.45	1.00	0.62	
Alibaba	Products	1.00	0.89	0.94	22.23	1.00	1.00	1.00	4.36	1.00	1.00	1.00	20057.52
Target	Products	1.00	1.00	1.00	47.50	1.00	1.00	1.00	12.77	0.98	1.00	0.99	6203.02
TDG Scholar	Authors	1.00	1.00	1.00	6.03	1.00	1.00	1.00	1.38	0.86	1.00	0.92	5371.48
	Hosts	1.00	1.00	1.00		1.00	1.00	1.00		0.14	0.99	0.25	
	Papers	1.00	0.25	0.40		1.00	0.00	0.00		1.00	0.68	0.81	
Ms. Academic	Authors	1.00	0.85	0.92	7.89	1.00	0.96	0.98	2.83	0.74	1.00	0.85	3226.16
	Papers	1.00	0.98	0.99		1.00	0.50	0.67		0.24	1.00	0.39	
Google Scholar	Citations	1.00	1.00	1.00	5.28	1.00	1.00	1.00	1.23	1.00	1.00	1.00	22.72
Arxiv	Authors	1.00	1.00	1.00	20.81	1.00	0.25	0.40	9.53	0.73	0.18	0.29	3124.00
	Papers	1.00	1.00	1.00		1.00	0.00	0.00		0.17	0.18	0.17	
	Abstracts	1.00	0.88	0.94		1.00	0.92	0.96		0.10	0.19	0.13	
Livejournal	News	1.00	0.64	0.78	3.30	1.00	0.00	0.00	1.22	1.00	1.00	1.00	2234.27
Xing	User Profiles	1.00	1.00	1.00	2.08	1.00	1.00	1.00	0.77	1.00	1.00	1.00	294.36
Odesk	User Profiles	1.00	1.00	1.00	4.70	1.00	0.49	0.66	2.61	0.52	1.00	0.68	701.22
	Skills	1.00	1.00	1.00		1.00	1.00	1.00		0.42	1.00	0.59	
ArticlesBase	Authors	1.00	1.00	1.00	3.41	1.00	1.00	1.00	2.00	0.92	1.00	0.96	53.89
Freelancer	Projects	1.00	1.00	1.00	4.47	1.00	1.00	1.00	26.33	0.96	1.00	0.98	223.30
PlentyOfFish	User Profiles	1.00	1.00	1.00	6.16	1.00	0.85	0.92	0.25	1.00	1.00	1.00	3137.06
Slideshare	Files	0.92	1.00	0.96	3.00	1.00	1.00	1.00	1.30	0.92	1.00	0.96	82.94
Netlog	User Profiles	1.00	1.00	1.00	3.72	1.00	0.85	0.92	0.59	1.00	1.00	1.00	19.38
Drupal	Projects	1.00	0.68	0.81	3.14	1.00	0.00	0.00	1.14	0.60	1.00	0.75	901.63
	Authors	1.00	1.00	1.00		1.00	1.00	1.00		0.36	1.00	0.53	
Newegg	Products	1.00	1.00	1.00	47.38	1.00	0.91	0.95	3.02	1.00	1.00	1.00	1109.52
Overblog	Articles	1.00	0.78	0.88	0.66	1.00	1.00	1.00	0.53	0.92	1.00	0.96	92.77
Chip	News	1.00	1.00	1.00	6.75	1.00	0.98	0.99	2.48	1.00	1.00	1.00	4.47
Battle.net	Forum Posts	1.00	1.00	1.00	1.95	1.00	0.74	0.85	3.36	1.00	1.00	1.00	42.91
Fiverr	Ads	0.96	1.00	0.98	4.16	1.00	0.91	0.95	0.73	0.96	1.00	0.98	752.00
Fotolia	Photos	1.00	1.00	1.00	12.00	1.00	1.00	1.00	0.84	1.00	1.00	1.00	5722.91
People	Styles	1.00	0.70	0.74	4.25	1.00	0.00	0.00	3.77	0.50	0.83	0.62	1689.41
	Babies	1.00	0.69	0.73		1.00	0.00	0.00		0.50	0.87	0.64	
	Covers	1.00	1.00	1.00		1.00	0.33	0.50		0.43	1.00	0.60	
	Articles	1.00	0.84	0.84		1.00	1.00	1.00		0.57	1.00	0.73	
Indeed	Jobs	1.00	1.00	1.00	4.27	1.00	0.00	0.00	2.45	1.00	1.00	1.00	486.26
Gamefaqs	Boards	1.00	1.00	1.00	8.61	1.00	0.98	0.99	0.48	1.00	1.00	1.00	6077.89

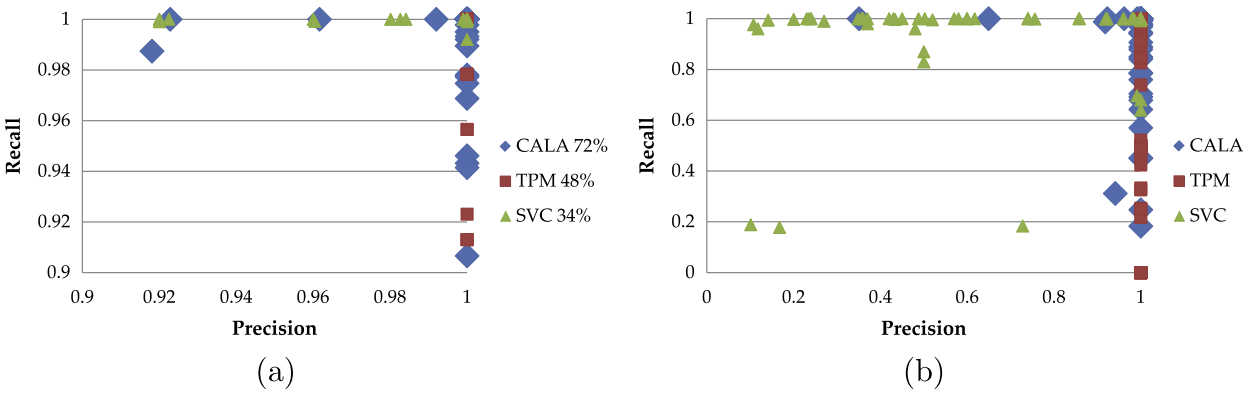

Fig. 6. Graphs showing the precision–recall of the three techniques.

$F1$; Fig. 6(a) illustrates this conclusion since the majority of points that correspond to CALA are very close to the upper right corner, whereas the points that correspond to the other techniques are more scattered. Fig. 6(b) provides a zoom on the right upper corner of the scatter plot ($P \geqslant 0.90, R \geqslant 0.90$). Note that 72% of the data points from CALA are located in this corner, whereas it only contains 48% of the data points from TPM and 34% of the data points from SVC. Note too that the results in Tables 4 and 5 support the idea that CALA is more effective regarding learning time than the other unsupervised technique (SVC), and that it is comparable to TPM which is supervised and requires to download a page before classifying it. Fig. 7(a) and (b) illustrates this idea. Note that times range from 0.66 to 47.50 s in the case of CALA, whereas they range from 0.17 to 26.33 in the case of TPM. On the contrary, learning times range from 4.47 to 49,386 s (more than 13 h) in the case of SVC. Therefore, CALA and TPM behave more homogeneously regarding learning times, and their learning times are significantly lower than those of SVC.

To confirm that the conclusions we have drawn from our empirical evaluation are valid, we need to perform a statistical ranking [21]. The first step is to determine if the evaluation results are normally distributed and have equal variances; in such a case, we must perform a parametrical analysis and in other case a non-parametrical analysis. We have conducted a Shapiro–Wilk test at the standard significance level $\alpha = 0.05$ on every measure and we have found out that none of them behaves normally. For instance, Shapiro–Wilk's statistic regarding the normality of CALA's precision is $W = 0.22$, whose p-value is 0.00; this is a strong indication that the data is not distributed normally. This is not surprising at all; a quick look at the scatter plot in Fig. 6(a) makes it clear that this cloud of points are far from a Gaussian circle.

As a conclusion, we have performed a non-parametric analysis, which consists of the following steps: (i) compute the rank of each technique from the evaluation data; (ii) determine if the differences in ranks are significant or not using Iman–Davenport's test; and (iii) if the differences are significant, then compute the statistical ranking using Bergmann–Hommel's test on every pair of techniques.

Table 6 presents the results of the analysis. Note that the p-value of Iman–Davenport's statistic is nearly zero in every case, which is a strong indication that there are statistically significant differences in the ranks we have computed from our experiments. It then proceeds to rank the techniques pairwise using Bergmann–Hommel's test. For the sake of readability, we also provide an explicit ranking in the last column. Note that our proposal ranks the first regarding $F1$, which means that it achieves a better trade-off between precision and recall than the other techniques. Regarding P and R the differences with TPM and SVC (which are the proposals that rank the first in each variable, respectively) do not seem to be statistically significant. Regarding efficiency, our proposal is a thousand orders of magnitude faster than SVC. Although it is slower than TPM, note that since TPM uses structure-based features, it has to download a page before classifying it. Therefore, although the learning time is faster, the classification

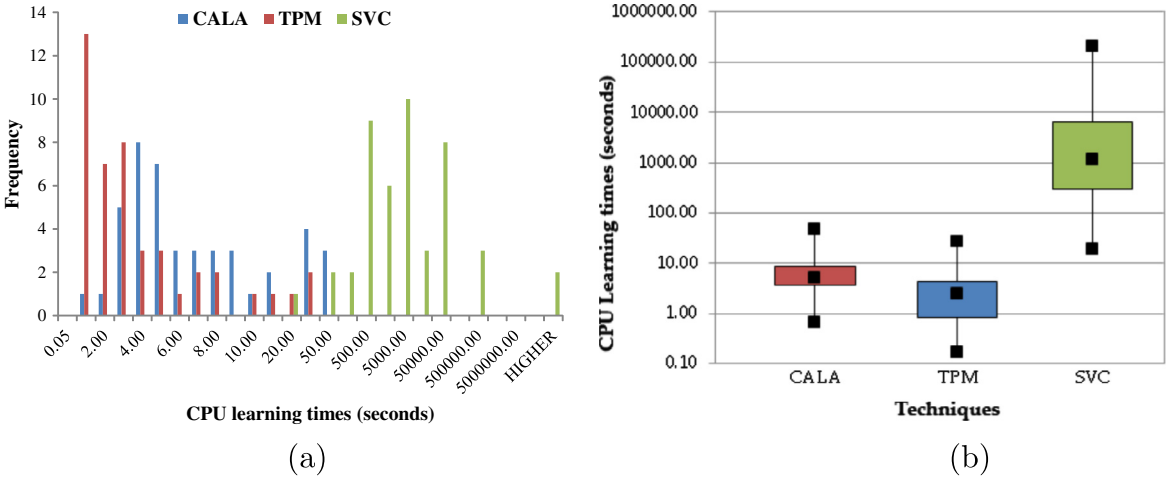

Fig. 7. CPU learning times of the three techniques: (a) Histogram and (b) Boxplot.

Table 6
Results of our statistical ranking.

Sample ranking		Iman–Davenport	Bergmann–Hommel's				Statistical ranking	
Tech	Rank	*p*-Value	Tech	CALA	TPM	SVC	Tech	Rank
P								
TPM	1.57	1.61E−16	CALA	–	4.73E−01	6.85E−10	CALA	1
CALA	1.69		TPM	–	–	1.70E−11	TPM	1
SVC	2.74		SVC	–	–	–	SVC	2
R								
SVC	1.64	1.42E−06	CALA	–	7.23E−04	1.39E−01	CALA	1
CALA	1.89		TPM	–	–	3.53E−06	SVC	1
TPM	2.46		SVC	–	–	–	TPM	2
F1								
TPM	1.54	4.69E−06	CALA	–	3.28E−04	1.84E−05	CALA	1
CALA	2.15		TPM	–	–	3.53E−01	TPM	2
SVC	2.31		SVC	–	–	–	SVC	2
T								
TPM	1.18	2.90E−33	CALA	–	1.99E−03	9.82E−08	TPM	1
CALA	1.84		TPM	–	–	1.12E−16	CALA	2
SVC	2.98		SVC	–	–	–	SVC	3

time is slower. As a conclusion, our experiments prove that there is enough statistical evidence to conclude that our proposal outperforms the others.

4.4. Corroboration of conjectures

In this section, we corroborate the conjectures on which our proposal relies.

Value of *M*: We conjectured that a relatively small number of hubs suffices to achieve a good effectiveness. Our experiments corroborate this hypothesis since examining more than 100 hubs did not result in better efficiency.
Value of *T*: We set a maximum number of attempts in Algorithm *gatherHubset* and conjectured that a relatively small number of attempts suffices to gather enough hubs. In our experiments, we managed to gather 100 hubs per site in a maximum of five attempts, which corroborates our conjecture.
Value of *N*: We conjectured that a relatively small number of keywords from the initial search page and subsequent hubs suffices to gather enough hubs. Our experiments corroborate this hypothesis since we managed to gather 100 hubs per site in every case by selecting only 10 words from each hub.
Distribution of *p*-estimators: We conjectured that the distribution of *p*-estimators has two peaks at 0.00 and 1.00. Fig. 8 depicts the distribution of *p*-estimators in our datasets; roughly 94.62% of the values range from 0.00 to 0.05, 1.63% range from 0.05 to 0.10 and 2.24% range from 0.95 to 1.00. Note that the graph is expressed in terms of a logarithmic scale.

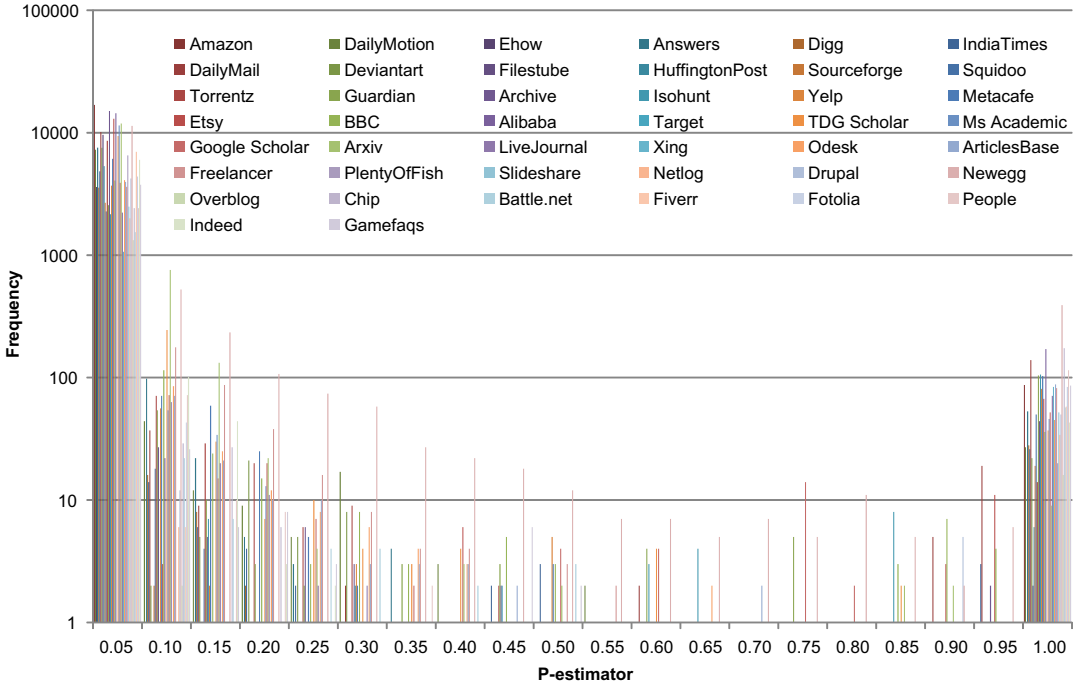

Fig. 8. Distribution of *p*-estimators in our experiments (logarithmic scale in the *Y* axis).

5. Conclusions

In this article, we have presented CALA, a proposal to automatically generate URL-based web page classifiers that can be used in the context of enterprise web information integration systems. Our proposal takes the URL of a web page with a keyword-based search form as input, and it outputs a set of patterns that represent the URLs of pages that belong to the same class. Our system fulfills three important requirements regarding enterprise web information integration: it performs a lightweight crawling to gather a sample of hubs automatically; it is unsupervised, since it learns a classifier from a training set of non-classified hubs that are gathered automatically from the web site, and it is based exclusively on URL features, so a page can be classified without downloading it previously.

We have validated our proposal using a collection of datasets that were gathered from 44 real-world web sites that we have made publicly available. We have built a set of patterns for each web site, and we have used them to classify further pages; we achieved an average precision of 98% and average recall of 90%. The times required to build those patterns were reasonable, similar to those of a supervised technique with which we compare ours. These results suggest that our proposal seems promising enough for real-world web page classification, that it is efficient in practice, that the patterns it builds are able to classify web pages accurately, and that it minimises the number of irrelevant pages that are downloaded, which makes it suitable for enterprise web information integration contexts.

Acknowledgements

This work was supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes (Grants TIN2007-64119, P07-TIC-2602, P08-TIC-4100, TIN2008-04718-E, TIN2010-21744, TIN2010-09809-E, TIN2010-10811-E, TIN2010-09988-E, and TIN2011-15497-E).

References

[1] Vinod Anupam, Juliana Freire, Bharat Kumar, Daniel F. Lieuwen, Automating web navigation with the WebVCR, Comput. Netw. 3 (1-6) (2000) 503–517, http://dx.doi.org/10.1016/S1389-1286(00)00073-6.

[2] Arvind Arasu, Hector Garcia-Molina, Extracting structured data from web pages, in: SIGMOD Conference, 2003, pp. 337–348, http://dx.doi.org/10.1145/872757.872799.

[3] Ziv Bar-Yossef, Idit Keidar, Uri Schonfeld, Do not crawl in the DUST: Different URLs with similar text, TWEB 3 (1) (2009) 3, http://dx.doi.org/10.1145/1462148.1462151.

[4] Ziv Bar-Yossef, Sridhar Rajagopalan, Template detection via data mining and its applications, in: WWW, 2002, pp. 580–591, http://dx.doi.org/10.1145/511446.511522.

[5] Eda Baykan, Monika Henzinger, Ludmila Marian, Ingmar Weber, A comprehensive study of features and algorithms for URL-based topic classification, TWEB 5 (3) (2011) 15, http://dx.doi.org/10.1145/1993053.1993057.

[6] Eda Baykan, Monika Rauch Henzinger, Ludmila Marian, Ingmar Weber, Purely URL-based topic classification, in: WWW, 2009, pp. 1109–1110, http://dx.doi.org/10.1145/1526709.1526880.

[7] Florian Beil, Martin Ester, Xiaowei Xu, Frequent term-based text clustering, in: KDD, 2002, pp. 436–442, http://dx.doi.org/10.1145/775047.775110.

[8] Asa Ben-Hur, David Horn, Hava T. Siegelmann, Vladimir Vapnik, Support vector clustering, J. Mach. Learn. Res. 2 (2001) 125–137. doi: 10.1.1.22.6663.

[9] Adam L. Berger, Stephen Della Pietra, Vincent J. Della Pietra, A maximum entropy approach to natural language processing, Comput. Linguist. 22 (1) (1996) 39–71. doi: 10.1.1.103.7637.

[10] Tim Berners-Lee, Roy T. Fielding, Larry Masinter, Uniform Resource Identifier URI: Generic Syntax. RFC, The Internet Engineering Task Force, 2005. <http://www.ietf.org/rfc/rfc3986.txt>.

[11] Smriti Bhagat, Graham Cormode, Irina Rozenbaum, Applying link-based classification to label blogs, in: WebKDD/SNA-KDD, 2007, pp. 97–117, http://dx.doi.org/10.1007/978-3-642-00528-2_6.

[12] Lorenzo Blanco, Valter Crescenzi, Paolo Merialdo, Structure and semantics of data-intensive web pages: an experimental study on their relationships, J. UCS 14 (11) (2008) 1877–1892, http://dx.doi.org/10.3217/jucs-014-11-1877.

[13] Lorenzo Blanco, Nilesh Dalvi, Ashwin Machanavajjhala, Highly efficient algorithms for structural clustering of large websites, in: WWW, 2011, pp. 437–446, http://dx.doi.org/10.1145/1963405.1963468.

[14] Sergey Brin, Extracting patterns and relations from the World Wide Web, in: WebDB, 1998, pp. 172–183, http://dx.doi.org/10.1007/10704656_11.

[15] Chia-Hui Chang, Mohammed Kayed, Moheb R. Girgis, Khaled F. Shaalan, A survey of web information extraction systems, IEEE Trans. Knowl. Data Eng. 18 (10) (2006) 1411–1428, http://dx.doi.org/10.1109/TKDE.2006.152.

[16] Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo, RoadRunner: towards automatic data extraction from large web sites, in: VLDB, 2001, pp. 109–118, doi: 10.1.1.21.8672.

[17] Luis M. de Campos, Juan M. Fernández-Luna, Juan F. Huete, Alfonso E. Romero, Probabilistic methods for link-based classification, in: INEX, 2008, pp. 453–459, http://dx.doi.org/10.1007/978-3-642-03761-0_47.

[18] Davi de Castro Reis, Paulo Braz Golgher, Altigran Soares da Silva, Alberto H.F. Laender, Automatic web news extraction using tree edit distance, in: WWW, 2004, pp. 502–511, http://dx.doi.org/10.1145/988672.988740.

[19] Elisabetta Fersini, Enza Messina, Francesco Archetti, Enhancing web page classification through image-block importance analysis, Inform. Process. Manage. 44 (4) (2008) 1431–1447, http://dx.doi.org/10.1016/j.ipm.2007.11.003.

[20] Dennis Fetterly, Mark Manasse, Marc Najork, Janet L. Wiener, A large-scale study of the evolution of web pages, Softw. Pract. Exper. 34 (2) (2004) 213–237, http://dx.doi.org/10.1002/spe.577.

[21] Salvador García, Alberto Fernández, Julián Luengo, Francisco Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining, Inform. Sci. 180 (10) (2010) 2044–2064, http://dx.doi.org/10.1016/j.ins.2009.12.010.

[22] Lise Getoor, Eran Segal, Ben Taskar, Daphne Koller, Probabilistic models of text and link structure for hypertext classification, in: IJCAI Workshop on Text Learning: Beyond Supervision, 2001, pp. 321–374, doi: 10.1.1.157.54.

[23] Peter Grünwald, Advances in Minimum Description Length: Theory and Applications, MIT Press, 2005.

[24] Ian Hacking, An Introduction to Probability and Inductive Logic, Cambridge University Press, 2001.

[25] Alon Y. Halevy, Naveen Ashish, Dina Bitton, Michael J. Carey, Denise Draper, Jeff Pollock, Arnon Rosenthal, Vishal Sikka, Enterprise information integration: successes, challenges and controversies, in: SIGMOD Conference, 2005, pp. 778–787, http://dx.doi.org/10.1145/1066157.1066246.

[26] Bin He, Mitesh Patel, Zhen Zhang, Kevin Chen-Chuan Chang, Accessing the deep web, Commun. ACM 50 (5) (2007) 94–101, http://dx.doi.org/10.1145/1230819.1241670.

[27] Inma Hernández, Carlos R. Rivero, David Ruiz, Rafael Corchuelo, A statistical approach to URL-based web page clustering, in: WWW (Companion Volume), 2012, pp. 525–526, http://dx.doi.org/10.1145/2187980.2188109.

[28] Inma Hernández, Carlos R. Rivero, David Ruiz, Rafael Corchuelo, Towards discovering conceptual models behind web sites, in: ER, 2012, pp. 166–175, http://dx.doi.org/10.1007/978-3-642-34002-4_13.

[29] Andreas Hotho, Alexander Maedche, Steffen Staab, Ontology-based text document clustering, Ger. J. Artif. Intell. 16 (4) (2002) 48–54. doi: 10.1.1.14.8083.

[30] ISO/IEC 13568:2002. Z Formal Specification Notation: Syntax, Type System and Semantics, 2002.

[31] Min-Yen Kan, Hoang Oanh Nguyen Thi, Fast web page classification using URL features, in: CIKM, 2005, pp. 325–326, http://dx.doi.org/10.1145/1099554.1099649.

[32] Jon M. Kleinberg, Authoritative sources in a hyperlinked environment, J. ACM 46 (5) (1999) 604–632, http://dx.doi.org/10.1145/324133.324140.

[33] Wallace Koehler, An analysis of web page and web site constancy and permanence, JASIS 50 (2) (1999) 162–180. doi: 10.1002/(SICI)1097-4571(1999)50:2<162::AID-ASI7>3.0.CO;2-B.

[34] Hema Swetha Koppula, Krishna P. Leela, Amit Agarwal, Krishna Prasad Chitrapura, Sachin Garg, Amit Sasturkar, Learning URL patterns for webpage de-duplication, in: WSDM, 2010, pp. 381–390, http://dx.doi.org/10.1145/1718487.1718535.

[35] Milos Kovacevic, Michelangelo Diligenti, Marco Gori, Veljko M. Milutinovic, Recognition of common areas in a web page using visual information: a possible application in a page classification, in: IEEE International Conference on Data Mining, 2002, pp. 250–257, http://dx.doi.org/10.1109/ICDM.2002.1183910.

[36] Oh-Woog Kwon, Jong-Hyeok Lee, Text categorization based on k-nearest neighbor approach for web site classification, Inform. Process. Manage. 39 (1) (2003) 25–44, http://dx.doi.org/10.1016/S0306-4573(02)00022-5.

[37] Jayant Madhavan, Loredana Afanasiev, Lyublena Antova, Alon Y. Halevy, Harnessing the deep web: present and future, Syst. Res. 2 (2) (2009) 50–54. doi: 10.1.1.145.915?.

[38] Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, Alon Y. Halevy, Google's deep web crawl, PVLDB 1 (2) (2008) 1241–1252, http://dx.doi.org/10.1145/1454159.1454163.

[39] Ricard Marxer, Piotr Holonowicz, Hendrik Purwins, Amaury Hazan, Dynamical hierarchical self-organization of harmonic, motivic, and pitch categories, in: NIPS Music, Brain and Cognition Workshop, Vancouver, Canada, 2007.

[40] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, Timm Euler, YALE: rapid prototyping for complex data mining tasks, in: KDD, 2006, pp. 935–940, http://dx.doi.org/10.1145/1150402.1150531.

[41] Paula Montoto, Alberto Pan, Juan Raposo, Fernando Bellas, Javier Lopez, Web navigation sequences automation in modern web sites, in: DEXA, 2009, pp. 302–316, http://dx.doi.org/10.1007/978-3-642-03573-9_25.
[42] Paula Montoto, Alberto Pan, Juan Raposo, José Losada, Fernando Bellas, Victor Carneiro, A workflow language for web automation, J. UCS 14 (11) (2008) 1838–1856, http://dx.doi.org/10.3217/jucs-014-11-1838.
[43] Kamal Nigam, Andrew Kachites Mccallum, Sebastian Thrun, Tom Mitchell, Text classification from labeled and unlabeled documents using EM, Mach. Learn. 39 (2–3) (1999) 103–134, http://dx.doi.org/10.1023/A:1007692713085.
[44] Gautam Pant, Padmini Srinivasan, Learning to crawl: comparing classification schemes, ACM Trans. Inform. Syst. 23 (4) (2005) 430–462, http://dx.doi.org/10.1145/1095872.1095875.
[45] Xiaoguang Qi, Brian D. Davison, Web page classification: features and algorithms, ACM Comput. Surv. 41 (2) (2009), http://dx.doi.org/10.1145/1459352.1459357.
[46] Sunita Sarawagi, Information extraction, Found. Trends Databases 1 (3) (2008) 261–377, http://dx.doi.org/10.1561/1900000003.
[47] Ali Selamat, Sigeru Omatu, Web page feature selection and classification using neural networks, Inform. Sci. 158 (2004) 69–88, http://dx.doi.org/10.1016/j.ins.2003.03.003.
[48] Lawrence Kai Shih, David R. Karger, Using URLs and table layout for web classification tasks, in: WWW, 2004, pp. 193–202, http://dx.doi.org/10.1145/988672.988699.
[49] Hassan A. Sleiman, Rafael Corchuelo, A survey on region extractors from web documents, IEEE Trans. Knowl. Data Eng. (2012), http://dx.doi.org/10.1109/TKDE.2012.135.TBP.
[50] Hassan A. Sleiman, Rafael Corchuelo, Trinity: on using trinary trees for unsupervised web data extraction, IEEE Trans. Knowl. Data Eng. (99) (2013), http://dx.doi.org/10.1109/TKDE.2013.161. 1-1, Preprint, ISSN 1041-4347.
[51] Jordi Turmo, Alicia Ageno, Neus Catalá, Adaptive information extraction, ACM Comput. Surv. 38 (2) (2006) 4, http://dx.doi.org/10.1145/1132956.1132957.
[52] Márcio L.A. Vidal, Altigran Soares da Silva, Edleno Silva de Moura, João M.B. Cavalcanti, Structure-based crawling in the Hidden Web, J. UCS 14 (11) (2008) 1857–1876, http://dx.doi.org/10.3217/jucs-014-11-1857.
[53] Karane Vieira, Altigran Soares da Silva, Nick Pinto, Edleno Silva de Moura, João M.B. Cavalcanti, Juliana Freire, A fast and robust method for web page template detection and removal, in: CIKM, 2006, pp. 258–267, http://dx.doi.org/10.1145/1183614.1183654.
[54] Wei Xie, Musa A. Mammadov, John Yearwood, Using links to aid web classification, in: ACIS-ICIS, 2007, pp. 981–986, http://dx.doi.org/10.1109/ICIS.2007.191.
[55] Jianping Zhang, Jason Qin, Qiuming Yan, The role of URLs in objectionable web content categorization, in: Web Intelligence, 2006, pp. 277–283, http://dx.doi.org/10.1109/WI.2006.170.
[56] Mingliang Zhu, Weiming Hu, Ou Wu, Xi Li, Xiaoqin Zhang, User oriented link function classification, in: WWW, 2008, pp. 1191–1192, http://dx.doi.org/10.1145/1367497.1367721.
[57] Shenghuo Zhu, Kai Yu, Yun Chi, Yihong Gong, Combining content and link for classification using matrix factorization, in: Research and Development in Information Retrieval, 2007, pp. 487–494, http://dx.doi.org/10.1145/1277741.1277825.